IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2019, No. 4, pp. 81–118. DOI:10.13154/tosc.v2019.i4.81-118

INT-RUP Secure Lightweight Parallel AE Modes

Avik Chakraborti[1], Nilanjan Datta[2], Ashwin Jha[2], Cuauhtemoc Mancillas-López[3], Mridul Nandi[2] and Yu Sasaki[1]

[1] NTT Secure Platform Laboratories, Tokyo, Japan
[2] Indian Statistical Institute, Kolkata, India
[3] Department of Computer Science, CINVESTAV-IPN, México City, Mexico
avikchkrbrti@gmail.com,nilanjan_isi_jrf@yahoo.com,ashwin.jha1991@gmail.com, cuauhtemoc.mancillas@cinvestav.mx,mridul.nandi@gmail.com,yu.sasaki.sk@hco.ntt.co.jp

Abstract. Owing to the growing demand for lightweight cryptographic solutions, NIST has initiated a standardization process for lightweight cryptographic algorithms. Specific to authenticated encryption (AE), the NIST draft demands that the scheme should have one *primary* member that has key length of 128 bits, and it should be secure for at least $2^{50} - 1$ byte queries and 2^{112} computations. Popular (lightweight) modes, such as OCB, OTR, CLOC, SILC, JAMBU, COFB, SAEB, Beetle, SUNDAE etc., require at least 128-bit primitives to meet the NIST criteria, as all of them are just birthday bound secure. Furthermore, most of them are sequential, and they either use a two pass mode or they do not offer any security when the adversary has access to unverified plaintext (RUP model). In this paper, we propose two new designs for lightweight AE modes, called LOCUS and LOTUS, structurally similar to OCB and OTR, respectively. These modes achieve notably higher AE security bounds with lighter primitives (only a 64-bit tweakable block cipher). Especially, they satisfy the NIST requirements: secure as long as the data complexity is less than 2^{64} bytes and time complexity is less than 2^{128}, even when instantiated with a primitive with 64-bit block and 128-bit key. Both these modes are fully parallelizable and provide full integrity security under the RUP model. We use TweGIFT-64[4,16,16,4] (also referred as TweGIFT-64), a tweakable variant of the GIFT block cipher, to instantiate our AE modes. TweGIFT-64-LOCUS and TweGIFT-64-LOTUS are significantly light in hardware implementation. To justify, we provide our FPGA based implementation results, which demonstrate that TweGIFT-64-LOCUS consumes only 257 slices and 690 LUTs, while TweGIFT-64-LOTUS consumes only 255 slices and 664 LUTs.

Keywords: OCB, OTR, TweGIFT, lightweight, INT-RUP, elastic-tweak

1 Introduction

Lightweight cryptography, that aims towards applications in resource constrained environments has seen a sudden surge in interest due to the advent of Internet of things (IoT). Particularly, lightweight authenticated encryption (AE) schemes are of utmost importance in establishing private and authenticated communication channels in IoT applications. This importance was addressed by recently concluded CAESAR competition [CAE14] and the ongoing NIST lightweight cryptography project [MBTM17]. In many of these designs, the internal state size reduction is the main priority. In this context, permutation-based schemes [BDPA11, CDNY18] have an advantage over block cipher-based schemes [CIMN17], as they do not need to store the key. However, to achieve comparable security, in general, the permutation size has to be almost similar to the block cipher size (key size + block size). In this work, we mainly focus on (tweakable) block cipher-based AE schemes.

Received: 2019-06-01, Revised: 2019-09-01, Accepted: 2019-11-01, Published: 2020-01-31

1.1 The NIST Lightweight Cryptography Standardization Project

The NIST project [MBTM17] has received submissions of lightweight designs for standardization. NIST set the following minimum requirements from the submissions.

- The key size should be at least 128 bits.
- When the key size is 128 (resp. 256) bits, any cryptanalytic attack should need at least 2^{112} (resp. 2^{224}) computations in a single key classical setting (i.e, the time complexity).
- There should be one primary recommendation for the scheme with key size at least 128 bits, nonce size at least 96 bits, tag length at least 64 bits and the total number of message bytes under a single key at least $2^{50} - 1$ (i.e, the data complexity).

In summary, the primary version of the authenticated encryption scheme should have security up to 2^{50} bytes of data and 2^{112} computations.

1.2 State of the Art on AE Modes in light of NIST Requirements

Depending on the performance requirements, AE modes can be categorized into two main structures.

Parallel modes: Parallel AE modes, such as OCB [KR16] and OTR [Min16], are designed to exploit the parallel computation infrastructure available in many high performance computing environments. These designs primarily focus on software efficiency with a reasonably fast implementation in hardware. Both OCB and OTR are efficient, rate[1] 1 and parallelizable. However, they require a relatively large state size[2] of $3n + \kappa$ and $4n + \kappa$ bits respectively, where n and κ are the block size and the key size respectively. Furthermore, they are only birthday bound secure in block size, which means they need at least 128-bit block cipher in order to satisfy the NIST criteria. This causes possible unsuitability of these designs in lightweight applications. In addition, they do not provide any security when the adversary gets access to unverified plaintext, i.e., the so-called RUP model [FJMV03, ABL+14], which might be relevant in many practical lightweight applications due to lack of memory buffer. In [ZWH17], Zhang et al. proposed a variant of OCB, called OCB-IC, which achieves integrity under RUP or INT-RUP security at the cost of making the construction rate 1/2. In [Nai17], Naito proposed another variant of OCB, called ΘCB3†, which offers beyond the birthday bound security.

Sequential modes: Sequential AE modes based on block cipher target minimal hardware implementation cost. Iwata et al. in [IMG+16] proposed two rate 1/2 AE modes CLOC and SILC to have a low hardware implementation with a state size of $(2n + \kappa)$ bits. Later, Wu et al. in [WH16] proposed a rate 1/2 AE mode JAMBU with even a lower state size of $(1.5n + \kappa)$ bits. In 2017, Chakraborti et al. proposed a combined feedback based AE mode COFB [CIMN17] that can be implemented with the same state size as JAMBU but can achieve the optimal rate 1. Recently proposed rate 1/2 AE modes SAEB [NMSS18] and SUNDAE [BBLT18] further optimizes the state size of $(n + k)$ bits. We note that most of these constructions do not have RUP security. However, SUNDAE achieves nonce-misuse security but SAEB does not.

All the above mentioned AE modes have birthday bound security on the primitive size. Consequently, in light of NIST security requirements, all the above mentioned modes require 128-bit block cipher with key size of approx 128-bit. A 128-bit block cipher like AES might not be well-suited for lightweight implementations. In fact, [BMR+13, BBM15] show that lightweight implementations (see [MPL+11]) of AES require much higher clock cycles,

[1] By rate, we mean number of message blocks processed per primitive invocation.
[2] Here state size means a theoretical estimation of the main registers.

when implemented in a small and serialized core. This is not desirable when throughput or energy consumption is also a concern in addition to the hardware footprint. On the other hand, 64-bit block ciphers such as PRESENT [BKL+07], SKINNY [BJK+16] or GIFT [BPP+17], have ultra lightweight implementation cost with a comparable throughput. This immediately raises an interesting problem:

(a) *Can we design an AE mode with a* 64*-bit block cipher (and* 128*-bit key) satisfying NIST Project's security requirements?*

1.3 Design Goals

With problem (a) in mind, we aim to design an algorithm that satisfies the following criteria:

- *Low State Size*: The overall state size of the construction should be as low as possible.
- *High Security*: The security of the mode should be high (preferably full security in block size) so that even 64-bit block size provides the required security.
- *Integrity Security under RUP (or INT-RUP security)*: The mode should provide integrity even in scenarios when unverified plaintexts are released. INT-RUP security is particularly significant in lightweight applications (smart-cards, RFID tags), where often the memory buffer is quite limited. In addition, INT-RUP is also useful in real-time streaming protocols (e.g. SRTP, SRTCP and SSH), where block-wise encryption/decryption is required and ciphertext/plaintext are released on-the-fly (though the verification oracle is also available to the attacker in addition to the unverified decryption oracle) in order to reduce the end-to-end latency.
- *Versatility*: The mode should also aspire to be flexible in its domain of applications, covering the spectrum of resource constrained devices.

Both OCB and OTR can achieve low state size and versatility using lighter primitives such as a 64-bit block cipher. However, as mentioned before, they do not achieve the desired security level when implemented with a 64-bit block cipher. This leads to another natural question:

(b) *Can we uplift the security level of* OCB *and* OTR *by keeping the functionalities as intact as possible?*

In fact a positive answer to (b) leads to a positive answer to (a). In general, this should result in a highly secure, efficient and significantly lighter design.

1.4 Our Contributions

The contributions of this paper are threefold:

1. LOTUS and LOCUS: We propose two new highly secure and hardware efficient block cipher based authenticated encryption modes, named LOCUS (Lightweight OCb with rUp Security) and LOTUS (Lightweight OTr with rUp Security) (see Sect. 3.1). Our new AE modes have the following features:

   (a) **High Security.** Both LOTUS and LOCUS satisfy $DT = O(2^{n+\kappa})$, where D and T denote the query and time complexities, respectively. Here $D < 2^n$, and $T < 2^\kappa$ are obvious conditions. We provide rigorous proofs (see Sect. 6) to obtain the respective security bounds in the ideal cipher model.

(b) **Lightweight.** The improved security ensures that both of these modes are secure with a 64-bit block cipher and 128-bit key. This essentially makes the modes lighter. Overall, LOTUS and LOCUS require only 388-bit and 324-bit states, respectively, which are much less than comparably secure OTR and OCB modes, i.e. 640-bit and 512-bit, respectively (using a 128-bit block cipher).

(c) **Efficient and Fast.** Our modes keep the basic features of OCB and OTR intact. Both are online, single pass and fully parallelizable.

(d) **INT-RUP Secure.** We show that LOTUS and LOCUS have full 64-bit INT-RUP security in the ideal cipher model, whereas it is well-known that the integrity of both OCB and OTR can be trivially broken in the RUP setting. We remark here that LOTUS and LOCUS do not achieve privacy notion in the RUP setting, (IND-CPA + PA1) of [ABL+14]. However, they achieve full 64-bit security in the usual notion of privacy (see Sect. 6).

2. We choose the recently proposed short-tweak tweakable block ciphers TweGIFT-64[4,16,16,4] [CDJ+19b] having a 64-bit block, a 128-bit key and only a 4-bit tweak to instantiate both LOTUS and LOCUS. Sometimes we use TweGIFT-64 as a short hand for TweGIFT-64[4,16,16,4]. As we use the concept of re-keying, we provide a comprehensive related-key analysis (see Sect. 4.2) of TweGIFT-64.

3. Finally, we implement both LOTUS and LOCUS with TweGIFT-64 as the underlying block cipher (see Sect. 5). We provide hardware implementation details on FPGA platform. We observe that our implementations achieve highly competitive result. LOCUS achieves a very low hardware area of only 257 slices and 690 LUTs, while LOTUS achieves an even lower hardware area of 255 slices and 664 LUTs. We also provide a benchmark on FPGA platform with several state of the art schemes containing lightweight designs.

We would like to point out that the proposed modes are well-suited for protocols that require both lightweight and high performance implementations e.g, lightweight clients interacting with high performance servers (e.g, LwM2M protocols [OS19]). Some of the existing sequential modes like sponges, SAEB are better in terms of area-efficiency, however, due to the sequential nature of such modes, they cannot utilize the parallel computing capability in high performance devices. On the contrary, our proposed modes are inherently parallel and can be implemented in fully pipelined manner keeping a comparable area-efficient implementation. Moreover, our modes have the lowest implementation area among all the existing parallel modes with RUP security.

1.5 Design Comparison

Table 1 summarizes a comparative study of our modes with popular lightweight AE modes. The underlying primitive sizes are chosen appropriately for each of them to satisfy the minimum security requirements by NIST. Note that in state size, we only count main registers and provide a theoretical estimation. Actual implementation may add some additional states required for the control unit and others. However, these additional states should be small when compared to the main register size. The security proofs for OCB, OTR, COFB, SAEB, SUNDAE are given in the standard model. Since they are all birthday bound secure in terms of the block size, they will achieve the same security level even in ICM.

Table 1: Comparison of various Lightweight AE modes with LOTUS and LOCUS. The block sizes are chosen such that they satisfy the criteria of NIST. The key size is 128-bit in all the cases.

Mode	State size	Primitive	Single Pass	Parallel	Rate	Inv-free	INT-RUP
OCB	512	128 (BC)	✓	✓	1	×	×
OTR	640	128 (BC)	✓	✓	1	✓	×
OCB-IC	512	128 (TBC)	✓	✓	1/2	×	✓
COFB	320	128 (BC)	✓	×	1	✓	×
SAEB	256	128 (BC)	✓	×	1/2	✓	–
SUNDAE	256	128 (BC)	×	×	1/2	✓	×
LOCUS	324	64 (TBC)	✓	✓	1/2	×	✓
LOTUS	388	64 (TBC)	✓	✓	1/2	✓	✓

1.6 Novelty of LOTUS and LOCUS

LOTUS and LOCUS introduce significant optimizations over OTR [Min16] and OCB [KR16] designs, respectively. Some of the novelties in LOTUS and LOCUS as compared to OTR and OCB are listed below:

1. LOTUS and LOCUS employ nonce-based key derivation as well as re-keying technique to ensure higher security with lighter primitives. As mentioned above, this allows for the use of ultralight 64-bit block ciphers.

2. A subtle change in the tag generation process saves additional n-bit state as compared to OTR and OCB, where n denotes the block size. So even with similar primitive size LOTUS and LOCUS would have drastically low hardware footprint as compared to OTR and OCB.

3. The nonce-based key derivation is also quite lightweight. The nonce-based key is simply the XOR of current nonce value and the master key. This helps in minimizing the latency as well as hardware circuitry overhead.

4. The simplicity in design permits simplified and shorter security proofs. We note that this is true for OTR and OCB as well, though they satisfy a weaker notion of integrity.

At a very high level, LOCUS can be viewed as an amalgamation of OCB-IC [ZWH17] and ΘCB3† [Nai17]. However, LOCUS improves over both of these designs on many fronts. In comparison to ΘCB3† it improves on two fronts. First, ΘCB3† requires a pseudorandom function call for nonce-dependent key generation, whereas nonce-based key derivation in LOCUS is much simpler. Second, ΘCB3† security bound contains a $DT/2^\kappa$ term, whereas LOCUS security bound is $DT/2^{n+\kappa}$. The bound for ΘCB3† is clearly not enough for NIST Project, when $\kappa = 128$. Although our technique for obtaining INT-RUP security is quite similar to OCB-IC, we emphasize that LOCUS achieves full n-bit INT-RUP security, whereas OCB-IC has just $n/2$-bit INT-RUP security.

1.7 Security Proof: Ideal Cipher Model vs Standard Model

We use nonce-based re-keying to get beyond the birthday bound security and it is a standard practice to use ideal cipher model (ICM) in this scenario, as duly mentioned and used in [BHT18, Men17, BT16] etc. The primary reason for switching from standard related-key model (SRKM) to ICM is the lossyness of the generic standard-to-ideal reduction. In SRKM, as shown in [Men17], one can achieve related-key SPRP (RKSPRP) advantage of roughly $DT/2^\kappa$, using key recovery attack which is quite loose and will not meet NIST

primary version criteria for AE with $\kappa = 128$. We emphasize that in many cases, including ours, this loss is meaningless as this attack on internal block cipher will not work on the mode due to the secret masking of the input and output of the block cipher (see Sect. 6.5). Although ICM might give an optimistic bound, we think that it captures the possible attack strategies in a better way as compared to SRKM. It is commonly believed that the SRKM might be too pessimistic, as noted in [BHT18, Men17, GPR14, BKR98]. It might be possible that a hybrid notion such as masked RKSPRP [Men17] could avoid such loss. However, such exposition is out of scope for this work.

2 Preliminaries

For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$ and $(n] := [n] \cup \{0\}$. For a finite set $\mathcal{X}$, $\mathsf{X} \xleftarrow{\$} \mathcal{X}$ denotes the uniform at random sampling of X from $\mathcal{X}$. For $n \in \mathbb{N}$, we write $\{0,1\}^+$ and $\{0,1\}^n$ to denote the set of all non-empty binary strings, and the set of all n-bit binary strings, respectively. We write ϕ to denote the empty string, and $\{0,1\}^* = \{0,1\}^+ \cup \{\phi\}$.

For $X \in \{0,1\}^*$, $|X|$ denotes the length (number of the bits) of X, where $|\phi| = 0$ by convention. For any non-empty binary string X, $(X_k, \ldots, X_1) \xleftarrow{n} x$ denotes the n-bit block parsing of X, where $|X_i| = n$ for $1 \le i \le k-1$, and $1 \le |X_k| \le n$. For $A, B \in \{0,1\}^*$ and $|A| = |B|$, we write $A \oplus B$ to denote the bitwise XOR of A and B.

We sometime use the terms (*complete*) *blocks* for n-bit strings, and *partial blocks* for m-bit strings, where $m < n$. Throughout, we use the function ozs, defined by the mapping

$$\forall X \in \bigcup_{m=1}^{n} \{0,1\}^m, \quad X \mapsto \begin{cases} 0^{n-|X|-1} \| 1 \| X & \text{if } |X| < n, \\ X & \text{otherwise,} \end{cases}$$

as the padding rule to map partial blocks to complete blocks. Note that the mapping is injective over partial blocks. For any $X \in \{0,1\}^+$ and $1 \le i \le |X|$, x_i denotes the i-th bit of X. For any binary string X and an integer $i \le |X|$, $\lfloor X \rfloor_i$ returns the least significant i bits of X, i.e. $x_i \cdots x_1$. For any integer i $\langle i \rangle_n$ denotes the n-bit unsigned representation of i.

2.1 Finite Field Arithmetic

The set $\{0,1\}^\kappa$ can be viewed as the finite field $\mathbb{F}_{2^\kappa}$ consisting of 2^κ elements. We interchangeably think of an element $A \in \mathbb{F}_{2^\kappa}$ in any of the following ways: (i) as a κ-bit string $a_{\kappa-1} \ldots a_1 a_0 \in \{0,1\}^\kappa$; (ii) as a polynomial $A(x) = a_{\kappa-1}x^{\kappa-1} + a_{\kappa-2}x^{\kappa-2} + \cdots + a_1 x + a_0$ over the field $\mathbb{F}_2$; (iii) a non-negative integer $a < 2^\kappa$; (iv) an abstract element in the field. Addition in $\mathbb{F}_{2^\kappa}$ is just bitwise XOR of two κ-bit strings, and hence denoted by $\oplus$. $P(x)$ denotes the primitive polynomial used to represent the field $\mathbb{F}_{2^\kappa}$, and α denotes the primitive element in this representation. The multiplication of $A, B \in \mathbb{F}_{2^\kappa}$ is defined as $A \odot B := A(x) \cdot B(x) \pmod{P(x)}$, i.e. polynomial multiplication modulo $P(x)$ in $\mathbb{F}_2$. For $\kappa = 128$, we fix the primitive polynomial

$$P(x) = x^{128} + x^7 + x^2 + x + 1. \tag{1}$$

Then, α, the primitive element, is $2 \in \mathbb{F}_{128}$. Throughout we use $\alpha = 2$, and $1 + \alpha = 3$, and "α-multiplication" to denote the operation of field multiplication on some element[3] and α.

[3] The element will be clear from the context.

2.2 Tweakable Block cipher

For $n, \tau, \kappa \in \mathbb{N}$, $\widetilde{\mathsf{E}}$-$n/\tau/\kappa$ denotes a tweakable block cipher family $\widetilde{\mathsf{E}}$, parametrized by the block length n, tweak length τ, and key length κ. For $K \in \{0,1\}^\kappa$, $T \in \{0,1\}^\tau$, and $M \in \{0,1\}^n$, we use $\widetilde{\mathsf{E}}_K^T(M) := \widetilde{\mathsf{E}}(K,T,M)$ to denote the invocation of the encryption function of $\widetilde{\mathsf{E}}$ on input K, T, and M. The decryption function is analogously defined as $\widetilde{\mathsf{E}}_K^{-T}(M)$. In the special case where the tweak set is a singleton, the resulting tweakable block cipher $\widetilde{\mathsf{E}}$ is simply referred as a block cipher E. We fix positive even integers n and κ to denote the *block size* and *key size*, respectively, in bits.

2.3 Authenticated Encryption in the Ideal Cipher Model

An authenticated encryption (AE) is an integrated scheme that provides both privacy of a plaintext $M \in \{0,1\}^*$ and authenticity of M as well as associated data $A \in \{0,1\}^*$. Taking a nonce N (which is a value unique for each encryption) together with associated data A and plaintext M, the encryption function of AE, enc_K, produces a tagged-ciphertext (C,T) where $|C| = |M|$ and $|T| = t$. Typically, t is fixed and we assume $n = t$ throughout the paper. The corresponding decryption function, dec_K, takes (N,A,C,T) and returns a decrypted plaintext M when the authentication on (N,A,C,T) is successful, otherwise returns the atomic error symbol denoted by $\perp$.

In this paper we consider a variant of the decryption interface, due to the added capability of our AE schemes. The decryption interface provides two algorithms, a decryption function dec_K that takes (N,A,C) and returns a decrypted plaintext M irrespective of the authentication result (hence we drop the tag value), and a verification function ver_K that takes (N,A,C,T) and returns a decrypted plaintext M only when the authentication succeeds, otherwise it returns $\perp$.

2.4 Security Definitions

A distinguisher $\mathscr{A}$ is an algorithm that tries to distinguish between two oracles $\mathcal{O}_0$ and $\mathcal{O}_1$ via black box interaction with one of them. At the end of interaction it returns a bit $b \in \{0,1\}$. We write $\mathscr{A}^{\mathcal{O}} = b$ to denote the output of $\mathscr{A}$ at the end of its interaction with $\mathcal{O}$. In the context of this paper, we will be concerned with computationally unbounded and deterministic distinguishers $\mathscr{A}$. The distinguishing advantage of $\mathscr{A}$ against $\mathcal{O}_0$ and $\mathcal{O}_1$ is defined as

$$\Delta_{\mathscr{A}}[\mathcal{O}_0; \mathcal{O}_1] = \left|\Pr[\mathscr{A}^{\mathcal{O}_0} = 1] - \Pr[\mathscr{A}^{\mathcal{O}_1} = 1]\right|, \tag{2}$$

where the probabilities depend on the random coins of $\mathcal{O}_0$ and $\mathcal{O}_1$.

2.4.1 TSPRP Security in Ideal Cipher Model

Let $\mathsf{TPerms}(\{0,1\}^\tau, \{0,1\}^n)$ be the set of all tweakable permutations with τ-bit tweak and n-bit block. We write $\widetilde{\Pi} \leftarrow\!\!\$\ \mathsf{TPerms}(\{0,1\}^\tau, \{0,1\}^n)$ to denote a tweakable random permutation. A tweakable block cipher $\widetilde{\mathsf{E}}$ is called a tweakable ideal cipher if $\widetilde{\mathsf{E}}_K \leftarrow\!\!\$\ \mathsf{TPerms}(\{0,1\}^\tau, \{0,1\}^n)$ for all $K \in \{0,1\}^\kappa$, i.e., $\widetilde{\mathsf{E}}$ behaves as a tweakable random permutation for all keys. The TSPRP advantage of any distinguisher $\mathscr{A}$ against a tweakable block cipher $\widetilde{\mathsf{P}}$ built upon a tweakable ideal cipher $\widetilde{\mathsf{E}}$ and instantiated with a key $\mathsf{K} \leftarrow\!\!\$\ \{0,1\}^\kappa$ is defined as

$$\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{P}}}(\mathscr{A}) := \Delta_{\mathscr{A}}\left[(\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]^{\pm}_{\mathsf{K}}, \widetilde{\mathsf{E}}^{\pm}); (\widetilde{\Pi}^{\pm}, \widetilde{\mathsf{E}}^{\pm})\right]. \tag{3}$$

The TSPRP advantage of $\widetilde{\mathsf{P}}$, is defined as

$$\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{P}}}(q, q_p) := \max_{\mathscr{A}} \mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{P}}}(\mathscr{A}),$$

where the maximum is taken over all distinguisher $\mathscr{A}$ bounded by q $\widetilde{\mathsf{P}}$ queries and q_p $\widetilde{\mathsf{E}}$ queries. The TPRP security game is a weaker variant of TSPRP where the distinguisher is restricted from making any inverse queries to the tweakable block cipher $\widetilde{\mathsf{P}}$, i.e.

$$\mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{\mathsf{P}}}(\mathscr{A}) := \Delta_{\mathscr{A}}\left[(\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]_{\mathsf{K}}, \widetilde{\mathsf{E}}^{\pm}); (\widetilde{\Pi}, \widetilde{\mathsf{E}}^{\pm})\right].$$

It is easy to see that $\mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{\mathsf{P}}}(q, q_p) \leq \mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{P}}}(q, q_p)$.

2.4.2 Privacy Security in Ideal Cipher Model

Given a distinguisher $\mathscr{A}$, we define the privacy advantage of $\mathscr{A}$ against an AE scheme Θ in the ideal cipher model as

$$\mathbf{Adv}^{\mathsf{priv}}_{\Theta[\widetilde{\mathsf{E}}]}(\mathscr{A}) := \Delta_{\mathscr{A}}[(\Theta.\mathsf{enc}_{\mathsf{K}}, \widetilde{\mathsf{E}}^{\pm}); (\$_{\mathsf{e}}, \widetilde{\mathsf{E}}^{\pm})], \tag{4}$$

where $\$_{\mathsf{e}}$ returns a uniform random string of the same length as the output length of $\Theta.\mathsf{enc}_K$. The privacy advantage of Θ is defined as

$$\mathbf{Adv}^{\mathsf{priv}}_{\Theta[\widetilde{\mathsf{E}}]}(q_e, q_p, \sigma_e, q_p) := \max_{\mathscr{A}} \mathbf{Adv}^{\mathsf{priv}}_{\Theta[\widetilde{\mathsf{E}}]}(\mathscr{A}),$$

where the maximum is taken over all distinguishers making q_e queries to the encryption oracle with an aggregate of σ_e blocks and q_p many primitive (ideal cipher) queries.

2.4.3 INT-RUP Security in Ideal Cipher Model

We say that an adversary $\mathscr{A}$ forges an AE scheme Θ under RUP in the ideal cipher model if $\mathscr{A}$ is able to compute a tuple (N, A, C, T) satisfying $\Theta.\mathsf{ver}_K(N, A, C, T) \neq \perp$, without querying (N, A, M) to $\Theta.\mathsf{enc}_{\mathsf{K}}$ and receiving (C, T), i.e. (N, A, C, T) is a non-trivial forgery. In this case, a forger can make additional q_d RUP decryption queries of the form (N, A, C) with a total of σ_d blocks to the oracle $\Theta.\mathsf{dec}_K$, with no restriction on nonce repetitions, and receive the corresponding M. One can also view the forging game in an equivalent distinguishing game. Under this equivalent setting, the integrity under RUP advantage for any distinguisher $\mathscr{A}$ is defined as

$$\mathbf{Adv}^{\mathsf{int\text{-}rup}}_{\Theta}(\mathscr{A}) := \Delta_{\mathscr{A}}[(\Theta.\mathsf{enc}_{\mathsf{K}}, \Theta.\mathsf{dec}_{\mathsf{K}}, \Theta.\mathsf{ver}_{\mathsf{K}}, \widetilde{\mathsf{E}}^{\pm}); (\Theta.\mathsf{enc}_{\mathsf{K}}, \Theta.\mathsf{dec}_{\mathsf{K}}, \perp, \widetilde{\mathsf{E}}^{\pm})], \tag{5}$$

where $\perp$ denotes the degenerate oracle that always returns $\perp$ symbol. The integrity under RUP advantage of Θ is defined as

$$\mathbf{Adv}^{\mathsf{int\text{-}rup}}_{\Theta}(q_e, q_d, q_v, \sigma_e, \sigma_d, \sigma_v, q_p) := \max_{\mathscr{A}} \mathbf{Adv}^{\mathsf{int\text{-}rup}}_{\Theta}(\mathscr{A}),$$

where the maximum is taken over all distinguishers making q_e encryption queries with an aggregate of σ_e blocks, q_d RUP queries with an aggregate of σ_d blocks, q_v verification attempts with an aggregate of σ_v blocks, and q_p ideal cipher queries.

Throughout we write a $(q_e, q_d, q_v, \sigma_e, \sigma_d, \sigma_v, q_p)$-distinguisher to represent a distinguisher that makes q_e encryption queries with an aggregate of σ_e many blocks, q_d decryption queries with an aggregate of σ_d many blocks, q_v verification queries with an aggregate of σ_v many blocks, and q_p primitive queries. Similarly, we can define distinguisher with smaller or larger tuple of resources.

2.5 Coefficient-H Technique

We outline the coefficient-H technique developed by Patarin, which serves as a "systematic" tool to upper bound the distinguishing advantage of any deterministic and computationally

unbounded distinguisher $\mathscr{A}$ in distinguishing the real oracle $\mathcal{O}_1$ (construction of interest) from the ideal oracle $\mathcal{O}_0$ (idealized version). The collection of all the queries and responses that $\mathscr{A}$ made and received to and from the oracle, is called the *transcript* of $\mathscr{A}$, denoted as ω. Sometimes, we allow the oracle to release more internal information to $\mathscr{A}$ only after $\mathscr{A}$ completes all its queries and responses, but before it outputs its decision bit.

Let Λ_1 and Λ_0 denote the transcript random variable induced by the interaction of $\mathscr{A}$ with the real oracle and the ideal oracle respectively. The probability of realizing a transcript ω in the ideal oracle (i.e., $\Pr[\Lambda_0 = \omega]$) is called the *ideal interpolation probability.* Similarly, one can define the *real interpolation probability.* A transcript ω is said to be *attainable* with respect to $\mathscr{A}$ if the ideal interpolation probability is non-zero (i.e., $\Pr[\Lambda_0 = \omega] > 0$). We denote the set of all attainable transcripts by Ω. Following these notations, we state the main result of coefficient-H Technique in Theorem 1. The proof of this theorem can be found in [Vau03].

Theorem 1. *Suppose for some* $\Omega_{\mathtt{bad}} \subseteq \Omega$*, which we call the* bad *set of transcripts, the following conditions hold:*

1. $\Pr[\Lambda_0 \in \Omega_{\mathtt{bad}}] \leq \epsilon_1$,
2. *For any good transcript* $\omega \in \Omega \setminus \Omega_{\mathtt{bad}}$*, we have* $\Pr[\Lambda_1 = \omega)] \geq (1 - \epsilon_2) \cdot \Pr[\Lambda_0 = \omega]$.

Then, we have

$$\Delta_{\mathscr{A}}[\mathcal{O}_0; \mathcal{O}_1] \leq \epsilon_1 + \epsilon_2. \tag{6}$$

3 Specification

In this section, we present the specifications of LOTUS and LOCUS that use a 4-bit short tweak tweakable block cipher TweGIFT-64 [CDJ+19b]. We give a short description of this design in Sect. 4.1.

3.1 LOTUS and LOCUS Modes

The encryption algorithm of both LOTUS and LOCUS modes receives an encryption key $K \in \{0,1\}^\kappa$, a nonce $N \in \{0,1\}^\kappa$, associated data $A \in \{0,1\}^*$, and a message $M \in \{0,1\}^*$ as inputs, and returns a ciphertext $C \in \{0,1\}^{|M|}$, and a tag $T \in \{0,1\}^n$. The decryption algorithm receives a key $K \in \{0,1\}^\kappa$, a nonce $N \in \{0,1\}^\kappa$, associated data $A \in \{0,1\}^*$, a ciphertext $C \in \{0,1\}^*$, and a tag $T \in \{0,1\}^n$ as inputs, and returns the plaintext $M \in \{0,1\}^{|C|}$ corresponding to C, if T authenticates.

Both LOTUS and LOCUS operate on n-bit blocks and use a tweakable block cipher as the underlying primitive. Both the algorithms share a common initialization and associated data processing phase. During the initialization phase, the κ-bit nonce N is XORed with the κ-bit secret key K to generate a κ-bit nonce-dependent encryption key K_N. Then, an n-bit nonce-dependent masking key Δ_N is generated using double encrypting a fixed value (here we have used 0^n) with key K and K_N successively with TBC.

3.1.1 Associated Data Processing in LOTUS and LOCUS

For associated data processing, we parse the data into n-bit blocks and process them in a similar way as the hash layer of PMAC [Rog04]. To process associated data block, we first update the current key value via α-multiplication. Next, we XOR the block with Δ_N and encrypt the value using $\widetilde{\mathsf{E}}$ with the fixed tweak value 2 and the updated key K_N and finally accumulate the encrypted output by XORing it to the previous checksum value. If the final block is partial, we use the tweak value 3 to process the final block. We refer to the output of the associated data processing as the AD checksum. The complete description of the associated data processing is depicted in Fig. 1 and formally specified in Algorithm 1.

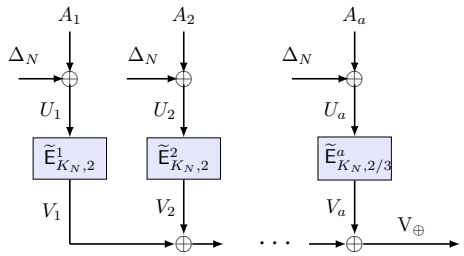

Figure 1: Associated Data Processing for both LOCUS and LOTUS. Here $\widetilde{\mathsf{E}}^i_{K_N,2}$ denotes invocation of $\widetilde{\mathsf{E}}$ with key $\alpha^i \odot K_N$ and tweak 0010. For the final associated data block, the use of $\widetilde{\mathsf{E}}^a_{K_N,2/3}$ indicates invocation of $\widetilde{\mathsf{E}}$ with key $\alpha^a \odot K_N$ and tweak 0010 or 0011 depending on whether the final block is full or partial.

3.1.2 Description of LOTUS

To process a message in LOTUS, we parse the data into $2n$-bit di-blocks and process them in a similar manner as OTR [Min16]. For each message di-block, we apply a simple variant of two-round Feistel cipher [LR85]. However, instead of one upper layer encryption and one lower layer encryption, here we use two successive encryptions in each layer. The intermediate states in between the encryptions in each layers are used to generate the checksum (that we call intermediate checksum), which helps in obtaining integrity security under RUP setting. To process a di-block, the key is first updated by an α-multiplication and the same key is used in the four tweakable block cipher calls. However, we use 4 different tweaks for the four calls (tweak 4 and 7 in the upper layer, and 5 and 8 in the lower layer) for the purpose of domain separation. Also, we use four different tweaks (12 and 14 in the upper layer and 13 and 15 in the lower layer) during the final di-block processing. The final di-block processing is slightly different and uses the length of the final di-block. To generate the tag, we apply XEX [Rog04] like transformation on the XOR of the intermediate checksum, AD checksum, and the final message block. The complete specification of LOTUS authenticated encryption is given in Algorithm 1. Figure 2 gives a pictorial description of the encryption process.

3.1.3 Description of LOCUS

To process a message in LOCUS, we parse the data into n-bit blocks and process them in a similar manner as OCB [RBB03]. For each of the message blocks, we first mask the block, then encrypt with the tweakable block cipher twice and then again mask to obtain the corresponding ciphertext block. Similar to LOTUS, the Δ_N masking is same along a query and the intermediate states (W_i in Fig. 3) between the two block cipher calls are XORed together to generate the intermediate checksum. For the last message block, instead of applying XEX on the message block, we apply it on the final block message length and XOR the output with the final message block. This strategy ensures identical processing for complete or incomplete final blocks. Again, similar to LOTUS, we update the key by α-multiplication before each block processing, and we use tweaks 4 and 12 in the upper and lower block cipher calls for non-final blocks, and tweaks 5 and 13 in the upper and lower block cipher calls for final blocks. The tag is generated identically to that of LOTUS. The complete specification of LOCUS authenticated encryption is given in Algorithm 2. The message processing part of the encryption algorithm is depicted in Fig. 3.

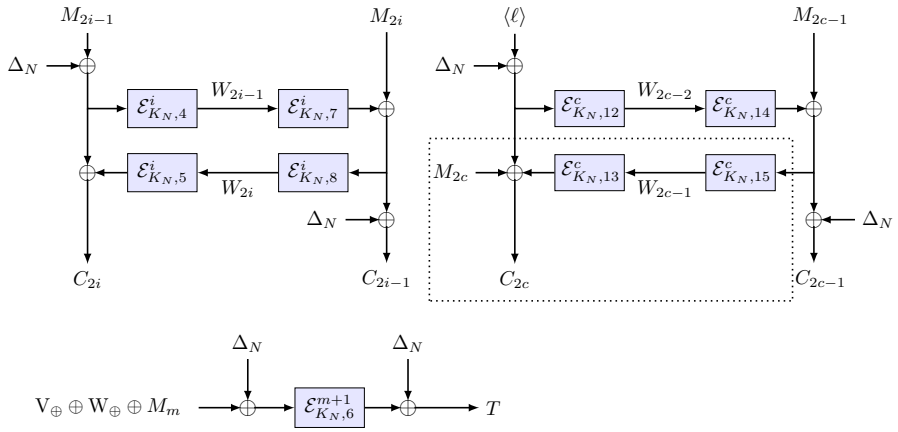

Figure 2: Processing of an m block message M and Tag Generation (assuming an odd number of input blocks) for LOTUS. The upper left part shows the message processing of an intermediate di-block and the upper right part depicts the message processing of the final di-block. The lower part shows the tag generation process. c denotes the number of di-blocks in the message i.e. $c = \lceil m/2 \rceil - 1$. The dotted part in the final di-block is executed only when the message has even number of blocks. We use the notation $\widetilde{\mathsf{E}}^i_{K_N,j}$ to denote invocation of $\widetilde{\mathsf{E}}$ with key $\alpha^{a+i} \odot K_N$ and tweak j, where a denotes the number of blocks of associated data corresponding to the message. Here $W_\oplus$ denotes the intermediate checksum value and $V_\oplus$ denotes the AD checksum value. $\langle \mathrm{len} \rangle_n$ is used to denote the n bit representation of the size of the final di-block in bits.

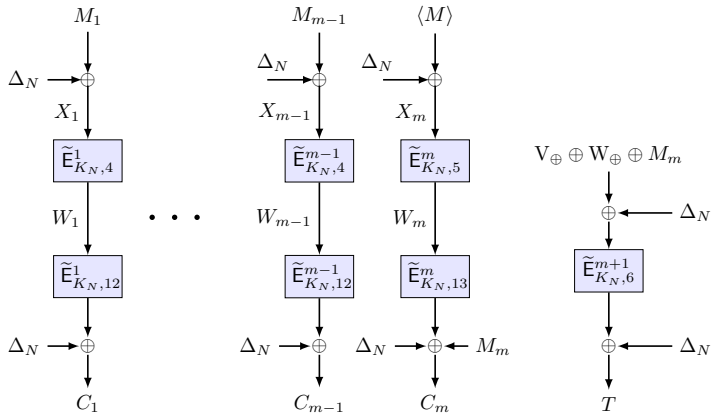

Figure 3: Processing of an m block message M and tag generation for LOCUS. $\langle \mathrm{len} \rangle_n$ is used to denote the n bit representation of the size of the final block in bits. $W_\oplus$ denotes the intermediate checksum value and $V_\oplus$ denotes the AD checksum value. $\widetilde{\mathsf{E}}^i_{K_N,j}$ is defined in a similar manner as in Fig. 2.

Algorithm 1 The encryption algorithm of LOTUS.

function LOTUS.enc(K, N, A, M)
  $C \leftarrow \perp, W_\oplus \leftarrow 0, V_\oplus \leftarrow 0$
  $(K_N, \Delta_N) \leftarrow \mathsf{init}(K, N)$
  if $|A| \neq 0$ **then**
    $(K_N, V_\oplus) \leftarrow \mathsf{proc\_ad}(K_N, \Delta_N, A)$
  if $|M| \neq 0$ **then**
    $(K_N, W_\oplus, C) \leftarrow \mathsf{proc\_pt}(K_N, \Delta_N, M)$
  $T \leftarrow \mathsf{proc\_tg}(K_N, \Delta_N, V_\oplus, W_\oplus)$
  return (C, T)

function init(K, N)
  $Y \leftarrow \widetilde{\mathsf{E}}^0_K(0^n)$
  $K_N \leftarrow K \oplus N$
  $\Delta_N \leftarrow \widetilde{\mathsf{E}}^1_{K_N}(Y)$
  return (K_N, Δ_N)

function proc_ad(K_N, Δ_N, A)
  $L \leftarrow K_N$
  $(A_1, \ldots, A_a) \xleftarrow{n} A$
  for $i = 1$ **to** $a - 1$ **do**
    $X \leftarrow A_i \oplus \Delta_N$
    $L \leftarrow L \odot \alpha$
    $V \leftarrow \widetilde{\mathsf{E}}^2_L(X)$
    $V_\oplus \leftarrow V_\oplus \oplus V$
  $X \leftarrow \mathsf{ozs}(A_a) \oplus \Delta_N$
  $L \leftarrow L \odot \alpha$
  if $|A_a| = n$ **then**
    $V \leftarrow \widetilde{\mathsf{E}}^2_L(X)$
  else
    $V \leftarrow \widetilde{\mathsf{E}}^3_L(X)$
  $V_\oplus \leftarrow V_\oplus \oplus V$
  return $(L, V_\oplus)$

function proc_pt(K_N, Δ_N, M)
  $L \leftarrow K_N$
  $(M_m, \ldots, M_1) \xleftarrow{n} M$
  $d = \lceil m/2 \rceil$
  for $i = 1$ **to** $d - 1$ **do**
    $j = 2i - 1$
    $X_1 \leftarrow M_j \oplus \Delta_N$
    $L \leftarrow L \odot \alpha$
    $W_1 \leftarrow \widetilde{\mathsf{E}}^4_L(X_1)$
    $Y_1 \leftarrow \widetilde{\mathsf{E}}^7_L(W_1)$
    $X_2 \leftarrow Y_1 \oplus M_{j+1}$
    $W_2 \leftarrow \widetilde{\mathsf{E}}^5_L(X_2)$
    $Y_2 \leftarrow \widetilde{\mathsf{E}}^8_L(W_2)$
    $W_\oplus \leftarrow W_\oplus \oplus W_1 \oplus W_2$
    $C_j \leftarrow X_2 \oplus \Delta_N$
    $C_{j+1} \leftarrow X_1 \oplus Y_2$
  $X_1 \leftarrow \langle |M| - 2(d-1)n \rangle_n \oplus \Delta_N$
  $L \leftarrow L \odot \alpha$
  $W_1 \leftarrow \widetilde{\mathsf{E}}^{12}_L(X_1)$
  $Y_1 \leftarrow \widetilde{\mathsf{E}}^{14}_L(W_1)$
  $X_2 \leftarrow Y_1 \oplus M_{2d-1}$
  $C_{2d-1} \leftarrow \lfloor X_2 \oplus \Delta_N \rfloor_{|M_{2d-1}|}$
  $W_\oplus \leftarrow W_\oplus \oplus W_1$
  $C \leftarrow (C_1, \ldots, C_{2d-1})$
  if $2d = m$ **then**
    $W_2 \leftarrow \widetilde{\mathsf{E}}^{13}_L(X_2)$
    $W_\oplus \leftarrow W_\oplus \oplus W_2$
    $Y_2 \leftarrow \widetilde{\mathsf{E}}^{15}_L(W_2)$
    $C_{2d} \leftarrow \lfloor X_1 \oplus Y_2 \rfloor_{|M_{2d}|} \oplus M_{2d}$
    $C \leftarrow C \| C_{2d}$
  $W_\oplus \leftarrow W_\oplus \oplus M_m$
  return $(L, W_\oplus, C)$

function proc_tg$(K_N, \Delta_N, V_\oplus, W_\oplus)$
  $L \leftarrow K_N \odot \alpha$
  if $(\lceil |A|/n \rceil + \lceil |M|/n \rceil) \mod 2 = 0$ **then**
    $X_\oplus \leftarrow V_\oplus \oplus W_\oplus \oplus \Delta_N$
  else
    $X_\oplus \leftarrow V_\oplus \oplus W_\oplus$
  $T \leftarrow \widetilde{\mathsf{E}}^6_L(X_\oplus) \oplus \Delta_N$
  return T

Algorithm 2 The encryption algorithm of LOCUS. The subroutines proc_ad and proc_tag are identical to the one used in LOTUS.

function LOCUS.enc(K, N, A, M)
  $C \leftarrow \perp,\ W_\oplus \leftarrow 0,\ V_\oplus \leftarrow 0$
  $(K_N, \Delta_N) \leftarrow \mathsf{init}(K, N)$
  if $|A| \neq 0$ **then**
    $(K_N, V_\oplus) \leftarrow \mathsf{proc\_ad}(K_N, \Delta_N, A)$
  if $|M| \neq 0$ **then**
    $(K_N, W_\oplus, C) \leftarrow \mathsf{proc\_pt}(K_N, \Delta_N, M)$
  $T \leftarrow \mathsf{proc\_tg}(K_N, \Delta_N, V_\oplus, W_\oplus)$
  return (C, T)

function proc_pt(K_N, Δ_N, M)
  $L \leftarrow K_N$
  $(M_1, \ldots, M_m) \xleftarrow{n} M$
  for $j = 1$ **to** $m - 1$ **do**
    $X \leftarrow M_j \oplus \Delta_N$
    $L \leftarrow L \odot \alpha$
    $W \leftarrow \widetilde{\mathsf{E}}^4_L(X)$
    $W_\oplus \leftarrow W_\oplus \oplus W$
    $Y \leftarrow \widetilde{\mathsf{E}}^{12}_L(W)$
    $C_j \leftarrow Y \oplus \Delta_N$
  $L \leftarrow L \odot \alpha$
  $X \leftarrow \langle |M_m| \rangle_n \oplus \Delta_N$
  $W \leftarrow \widetilde{\mathsf{E}}^5_L(X)$
  $Y \leftarrow \widetilde{\mathsf{E}}^{13}_L(W)$
  $C_m \leftarrow \lfloor Y \oplus \Delta_N \rfloor_{|M_m|} \oplus M_m$
  $W_\oplus \leftarrow W_\oplus \oplus W \oplus M_m$
  $C \leftarrow (C_1, \ldots, C_m)$
  return $(L, W_\oplus, C)$

3.2 Design Rationale

In this section, we briefly describe the various design choices and rationale for our proposals. Our primary goal is to design a lightweight AEAD that should be efficient, provides high performance and performs reasonably well in low-end devices as well. For efficiency, the AEAD should be one pass. To obtain high performance capability, we aim for parallelizability. In addition, we demand integrity in the RUP model. This is specially useful for memory-constrained lightweight applications.

We start with two well-known modes, namely OCB and OTR. Both OCB and OTR satisfy the first two properties. OCB is online, one-pass and parallelizable. OTR has all these features plus it offers inverse-freeness, albeit in exchange for a larger state (as it works on di-blocks). However, both of them are insecure under the RUP model. This motivates us to design an AE mode which is structurally as simple as OCB and OTR but achieves RUP security while keeping the primary features, such as efficiency and parallelism.

The new proposals LOTUS and LOCUS replace one block cipher call by two calls. The rationale behind this modification is the observation that the intermediate state between the two block cipher invocations can be used to generate a checksum, which is completely hidden and hence cannot be controlled by the adversary (even if the adversary is allowed to make RUP queries). This hidden checksum ensures integrity security in RUP model. The additional block cipher call per message block increases the number of block cipher calls from ℓ to $2\ell + 1$ to process an ℓ-block message. However, this is the minimum number of non-linear invocations used for any state-of-the-art INT-RUP secure parallel AEAD mode.

The associated data processing phase is based on a simple variant of the hash layer of PMAC, and the computation is completely parallel. The associated data processing can be done in parallel with the plaintext and/or ciphertext processing in order to maximize the performance in parallel computing environments.

Both OCB and OTR generate the tag using the checksum (simple XORs) of all the plaintext blocks and the output of the processed associated data. However, two separate states are required to hold the message checksum and the AD checksum. We obtain INT-RUP security, by using an intermediate checksum (hidden to the adversary) instead

of the plaintext checksum. Moreover, we do not store the intermediate checksum and AD checksum separately. Rather, we XOR the two checksums, which means that in a sequential implementations, the intermediate checksum can be computed on top of the AD checksum. This reduces the overall state size by size of one block.

A notable change in LOTUS and LOCUS is the use of nonce and position dependent keys. OCB and OTR have only birthday bound security on the block size. This is because the security is generally lost once the input/output of any two distinct block cipher calls matches, as the two calls share the same encryption key. In LOTUS and LOCUS, we overcome the birthday bound barrier by changing the key and tweak pair for each block cipher call. So even if there is a collision among inputs/outputs, the security remains intact, as the block cipher keys or tweaks are distinct. In fact, our modes are secure up to data complexity of 2^n, and time complexity of 2^κ, and combined data-time complexity up to $2^{n+\kappa}$. This, in turn, helps us to construct AEAD algorithms with the desired security level using an ultra-lightweight short tweak tweakable block cipher of size 64 bits.

Remark 1. We remark here that our specification of LOTUS and LOCUS deviates from the original definitions available in [CDJ+19a]. First, we use distinct tweaks in the ciphertext generation calls, i.e. 4 tweaks in LOTUS and 2 tweaks in LOCUS. Second, in the tag generation module, we perform the input masking by Δ_N only when the total input length (sum of associated data and message block length) is even. We have made these small modifications in the specification to simplify and modularize the proofs along the line of OCB [Rog04]. The security of the original constructions can be shown, via a dedicated proof, to be exactly the same. See Sect. 6 for more details.

Remark 2. OCB-IC by Zhang et al. [ZWH17] achieves INT-RUP security using a similar idea as ours (two calls in ciphertext generation). However, we improve on several fronts. We reduce the state size by avoiding the additional storage for AD checksum. Further, we improve the security from $n/2$-bit to n-bit.[4] This helps us in using a lighter primitive as compared to OCB-IC.

4 Instantiation

4.1 The TweGIFT-64 Tweakable Block Cipher

Here we briefly revisit the short tweak Tweakable block cipher TweGIFT-64, which is a 64-bit tweakable block cipher with 4-bit tweak and 128-bit key. It is identical to TweGIFT-64[4,16,16,4] [CDJ+19b]. However, for the sake of completeness, we briefly describe the tweakable block cipher. TweGIFT-64 uses 28 rounds, where each round consists following operations:

SubCells: TweGIFT-64 employs the same invertible 4-bit S-box as GIFT-64-128 and applies it to each nibble of the cipher state.

PermBits: TweGIFT-64 also uses the same bit permutation that was used in GIFT-64-128. The permutation maps bits from bit position i of the cipher state to bit position $GP(i)$, where

$$GP(i) = 4\lfloor i/16 \rfloor + 16\Big(\Big(3\lfloor (i \bmod 16)/4 \rfloor + (i \bmod 4)\Big) \bmod 4\Big) + (i \bmod 4).$$

AddRoundKey: In this step, a 32-bit round key is extracted from the master key state and added to the state (at bit positions i and $i+1$ for $i = 0, 1, \ldots, 15$). After that, the master key state is rotated by some bits. This operation is also identical to that of GIFT.

More precisely, the 128-bit key is loaded to eight 16-bit registers $k_0, \ldots, k_7$. Let U and V be two 16-bit registers to be extracted. $U\|V$ are $k_1\|k_0$, $k_3\|k_2$, $k_5\|k_4$ and $k_7\|k_6$ for

[4] The security of OCB-IC does not improve even in ideal cipher model.

round $4i$, $4i+1$, $4i+2$, and $4i+3$ for $i = 0, 1, \ldots, 6$, respectively. After $U\|V$ are XORed to the state, they are updated as follows: $U \leftarrow U \ggg 2, V \leftarrow V \ggg 12$.

AddRoundConstant: A single bit "1" and a 6-bit round constant are XORed into the cipher state at bit position 63, 23, 19, 15, 11, 7 and 3, respectively. The round constants are generated using the same 6-bit affine LFSR as SKINNY [BJK+16] and GIFT-64-128 [BPP+17].

AddTweak: For tweak processing, we first expand the 4-bit tweak into a 16-bit codeword using an efficient linear code. Let t_0, t_1, t_2, t_3 be 4 bits of the tweak and t_s be the sum of the 4 bits. Then we compute $t_{i+4} = t_i \oplus t_s$ for $i = 0, 1, 2, 3$ and $t_{i+8} = t_i$ for $i = 0, 1 \ldots, 7$. Then this expanded codeword $t_0, \ldots, t_{15}$ are XORed to the state (at bit position $i+3$ for $i = 0, 1, \ldots, 15$) at an interval of 4 rounds.

4.2 Security Analysis of TweGIFT-64

4.2.1 Intuition

Without exploiting the tweak, TweGIFT-64 is exactly the same as the original GIFT-64-128, which has already received several third-party security analysis. This principle can apply both in the single-key and the related-key settings. To exploit the features that do not exist for GIFT-64-128 but do for TweGIFT-64, attackers need to exploit the 4-bit tweak input. However, only with 4 additional bits, what attackers can do is very limited. In addition, the 4-bit tweak input is further expanded to 16 bits in an uncontrollable way. Thus, the best attack strategy is to attack the original GIFT-64-128 instead of trying to exploit the 4-bit tweak input.

One may consider using round keys to cancel the impact of the tweak because both of round keys and the expanded tweak are XORed to the state. In particular, for differential cryptanalysis, one may consider canceling the tweak difference by the round-key difference in the related-key setting. However, AddRoundConstant and AddTweak are designed to XOR the key bits and tweak bits to different bit positions. Thus, they cannot cancel each other. In the following, we discuss more details in the case of differential cryptanalysis.

4.2.2 Security against Differential Cryptanalysis

The exact security bound, e.g. the lower bound of the number of active S-boxes and the upper bound of the differential characteristic probability, can be obtained by using various tools based on MILP and SAT, however to derive such bounds for the entire construction with 128-bit key difference is often infeasible.

Here we focus on the feature that the tweak expansion function ensures that the number of active bits in the expanded tweak is at least 8 when the tweak difference is non-zero. This implies that differential trails with non-zero tweak difference will have a large number of active S-boxes around the tweak injection. This motivates us to evaluate the tight bound of the differential characteristic probability for the 2-round transformation followed by the tweak injection and another 2-round transformation, which we call "4-round core." Let p_{core} be the maximum differential characteristic probability of the 4-round core. Then, the probability for the entire construction is upper bounded by $(p_{\text{core}})^6$ because 28 rounds of TweGIFT-64 contain six 4-round cores (Fig. 4).

p_{core} in the single-key setting. As evaluated by [CDJ+19b], p_{core} can be evaluated by using the MILP based tool. p_{core} of the 4-round core is $2^{-25.6}$, hence the probability for the entire construction is upper bounded by $2^{-25.6\times 6} = 2^{-153.6}$. Because the block size of TweGIFT-64 is 64 bits, it well resists the differential cryptanalysis.

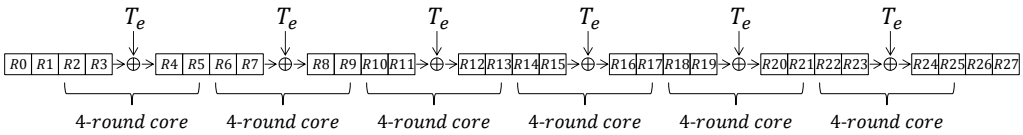

Figure 4: Six 4-round cores of TweGIFT-64. T_e denotes the 16-bit expanded tweak.

Table 2: Differential Trail with $p_{\text{core}} = 2^{-16}$ in the Related-Key Setting.

Round	Mask	Differential Mask	Key/Tweak Difference	Probability
1	Before SC	0000 0000 0000 0000		1
	After SC	0000 0000 0000 0000		
			Δkey: 0000 0000	
	Round key	0000 0000 0000 0000		
2	Before SC	0000 0000 0000 0000		1
	After SC	0000 0000 0000 0000		
			Δkey: 0000 0000	
	Round key	0000 0000 0000 0000		
			Δexpanded tweak: e1e1	
3	Before SC	8880 0008 8880 0008		2^{-16}
	After SC	3330 0003 3330 0003		
			Δkey: aaa5 aa5a	
	Round key	0000 0000 0000 0000		
4	Before SC	0000 0000 0000 0000		1
	After SC	0000 0000 0000 0000		
			Δkey: 0000 0000	
	Round key	0000 0000 0000 0000		

p_{core} in the related-key setting with non-zero tweak difference. We also used the MILP based tool to derive p_{core} by allowing the difference in the 128-bit key. Recall that the round key size is 32 bits, hence all subkey bits in the consecutive 4 rounds are independent. Under the condition that at least 1 tweak bit is active, it turned out that activating round keys leads to a higher p_{core} than the single-key setting. The results show that p_{core} in the related-key setting is 2^{-16}, hence the maximum differential characteristic probability of 28 rounds is upper bounded by $2^{-16\times 6} = 2^{-96}$. This can also be viewed that the maximum differential characteristic probability reaches $2^{-16\times 4} = 2^{-64}$ for 16 rounds and we have 12 rounds for a margin. One of the best related-key differential trails for the 4-round core is fully specified in Table 2.

We emphasize that the maximum differential probability of 4 rounds of the original GIFT-64-128 in the related-key setting is 1, namely 4 rounds can be bypassed without activating any S-box (This occurs when only the 4th round key is active). To attack TweGIFT-64 by exploiting the tweak is more difficult than to attack GIFT-64-128.

Gap between the lower and upper bounds. One may wonder if the trail in Table 2 is iterative and thus it immediately leads to the differential trail matching the upper bound of the characteristic probability, 2^{-96} for 28 rounds. This is because the input and output state difference in Table 2 is both 0 (iterative), the expanded 16-bit tweak is iterative, and the active two 16-bit key registers are always k_0 and k_1. However, we confirmed that TweGIFT-64 does not allow such simple iterative characteristics. Indeed, bit-rotation of the key registers for each round prevents from iterating the same differential propagation multiple times.

To demonstrate this intuition clearly, we checked the behavior of the differential trail in the subsequent 4-round core after the type of trails in Table 2. Namely, we first made a limitation that only the third round is active in the first 4-round core, and then to check the behavior of the second 4-round core. The results show that when only the third round in the second 4-round core is active, the highest probability of the two consecutive 4-round cores is 2^{-64} (2^{-32} per 4-round core). This occurs in the following configuration.

- 4 bits of the tweak is all active, which makes ΔT_e be `0xffff`.
- All 16 S-boxes have differential propagation `0x8` to `0x3` with probability 2^{-2}.
- All 16 bits of k_0 and k_1 are active, i.e. `0xffff`, which cancels the difference of the S-box output. (The difference `0xffff` is invariant for any rotation operation.)

Hence, to the best of our knowledge, the current lower bound of the differential characteristic probability for 28 rounds with non-zero tweak difference in the related-key setting is $2^{-32\times 6} = 2^{-192}$.

By not trying to cancel the difference in the second 4-round core, the third round of the second 4-round core can be bypassed with probability 2^{-16}. Hence starting from the first round, this will yield a differential characteristic with probability 2^{-32} up to 9 rounds. However, since the difference soon diffuses in an uncontrollable way, it is inevitable to activate more S-boxes for the entire construction with this approach.

Remarks on three 8-round cores. 28 rounds of TweGIFT-64 can also be viewed as containing three of the 8-round core by ignoring two AddTweak operations between two 8-round cores. We also evaluated the maximum differential characteristic probability of the 8-round core by using the MILP-based tool, which turned out to be $2^{-26.7}$. Hence, from this evaluation, the probability for the entire construction can be upper bounded only by $2^{3\times -26.7} = 2^{-80.1}$.

This observation demonstrates the difficulties of exploiting our tweak injection in another way. The difficulty of controlling differential trails lies in the heavy weight of the expanded tweak and thus to count as many tweak injection as possible would be the best to derive good bounds.

We notice that the maximum differential characteristic probability of 8 rounds of the original GIFT-64-128 in the related-key setting is 2^{-8}, which contains only 4 active S-boxes. The tweak expansion of TweGIFT-64 introduces many (at least 8) active S-boxes around the tweak injection, and this prevents the efficient differential trails available in GIFT-64-128.

Related-key security with zero tweak difference. Because of the difficulty of exploiting the tweak, zero-tweak difference would be the most natural scenario to attack TweGIFT-64. As discussed before, the related-key security of TweGIFT-64 without using tweak difference can be reduced to the related-key security of GIFT-64-128. Indeed, the keys are computed by a predictable way in the mode and used with a fixed tweak. This implies that related-key security of TweGIFT-64 matters in the related-key security of the entire construction.

At the time of the publication of GIFT-64-128 [BPP+17], the designers mentioned that *"GIFT aims at single-key security, so we do not claim any related-key security (even though no attack is known in this model as of today)."* On the other hand, several papers tried to attack GIFT-64-128 in the related-key setting, e.g. related-key boomerang attack on 23 rounds [LS19] and related-key rectangle attack on 23 rounds [CWZ19] and on 24 rounds [ZDM+19]. Without some innovation of the cryptanalytic technique, 28 rounds of GIFT-64-128 would resist those approaches in the related-key setting.

Regarding the related-key differential cryptanalysis of GIFT-64-128, one may expect that the above-mentioned 8-round characteristic with probability 2^{-8} can be iterated three times to derive $(2^{-8})^3 = 2^{-24}$ as the upper bound of the probability for 24 rounds. However this is only the loose bound, and such high probability characteristic in fact does not exist. To find high probability differential characteristics, to use automated tools such as SAT or MILP is a popular approach. In fact GIFT receives a lot of attention with this respect both in the single-key and related-key settings [ZDY18, ZZDX19, LS19, CWZ19, ZDM+19, JZD19]. In particular, Liu et al. evaluated the lower bound of the number of active S-boxes of GIFT-64-128 in the related-key setting up to 19 rounds [LS19, Table 4], which shows that

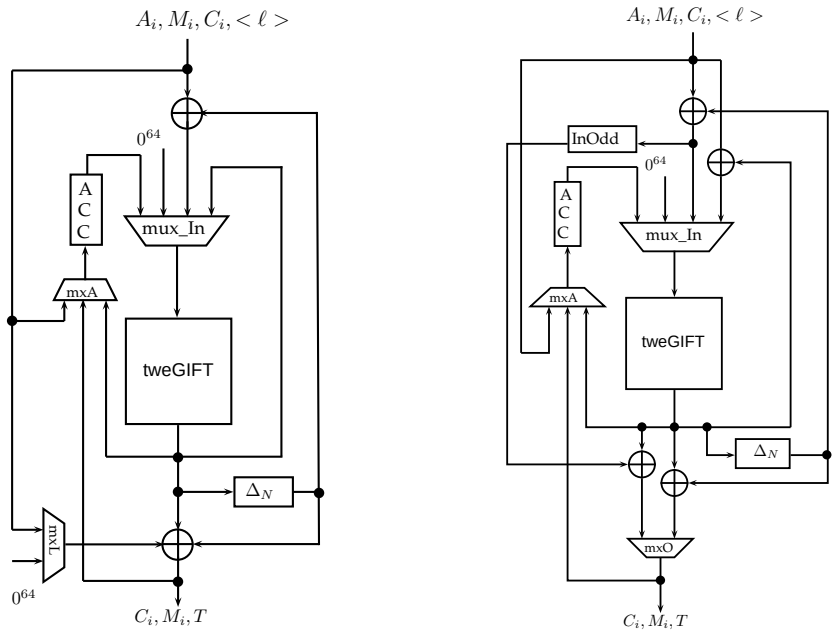

Figure 5: Hardware Architectures of LOCUS (left) and LOTUS (right)

the number of active S-boxes for r rounds is $2r-11$ for $r \geq 11$. Moreover, Liu et al. showed that the probability of the related-key differential characteristic is upper bounded by 2^{-79} (2^{80} paired queries) for 19 rounds [LS19, Table 5]. Considering that the attacker can append several rounds on top of the distinguisher, one may want to increase the number of rounds. The analysis to identify the number of added rounds to provide sufficient security margin is an open problem.

5 Hardware Implementation

In this section, we provide a brief idea on the FPGA implementations of our designs. All the hardware implementations are written in VHDL and are implemented on both Virtex 6 xc6vlx760 and Virtex 7 xc7vx415t using Xilin ISE 14.7 and Vivado 2018.3 respectively as implementation tool. In all the cases the optimization strategy is speed oriented.

5.1 Hardware Architecture

We implement combined encryption-decryption circuits for both the ciphers in a round-based architecture with 64 bit data path. Both the architectures are more or less similar with a few differences in the message processing phase. The main modules and the registers are briefly described below:

Registers. Both the architectures mainly contains four registers. A state register of 64 bits is used to store the encryption state. A key register of 128 bits is used to store the master secret key. Note that the key schedule of the underlying block cipher is palindromic and hence it removes the requirement to use a subkey register and only one subkey register suffices. A 64 bit checksum register is used to store and update the checksum value, and one Δ register is used to store the Δ_N value. The state register and key register are part of the module TweGIFT-64.

TweGIFT-64 Module. This module actually describes one round function of the TweGIFT-64 block cipher (Fig. 7). For LOCUS, we need both forward and inverse block cipher calls. Each round of a forward call consists of a sequence of S-box, bit-permutation and add round key operations. The inverse block cipher call performs add round key, inverse bit permutation and the inverse S-box operations (in the reverse direction). Internally a tweak value is added to the state register after each of the five consecutive rounds. For LOTUS, only the forward block cipher call is required. It takes 64 bit input from the state register computes one round of forward (or inverse in case of LOCUS) operations and then updates the state and send the output either to the accumulator or again to the TweGIFT-64 round module or release the output as the final tag (added to Δ_N before the tag release).

Accumulator Module. The accumulator module ACC computes the checksum value of the ECB layer and the last block to compute the tag.

InOdd Module. This module is specific LOTUS and it is used to detect whether the counter value for the current message block has an odd index. This is required as it processes two blocks at a time to xor it with the output of TweGIFT-64 to produce the output.

Finite State Machine. We also report the finite state machine (FSM) that controls the circuit flow by controlling and updating the internal signals and sending them to the internal modules. The FSM has a simple structure. The overall hardware architecture for both the designs is given in Fig. 5. This module is used to control the circuit. It is used to generate and send signals to the internal modules and the functionalities of the circuit are described by several states. Fig. 6 describes the state transitions of the finite state machine (FSM) for the AE designs. The states are described below.

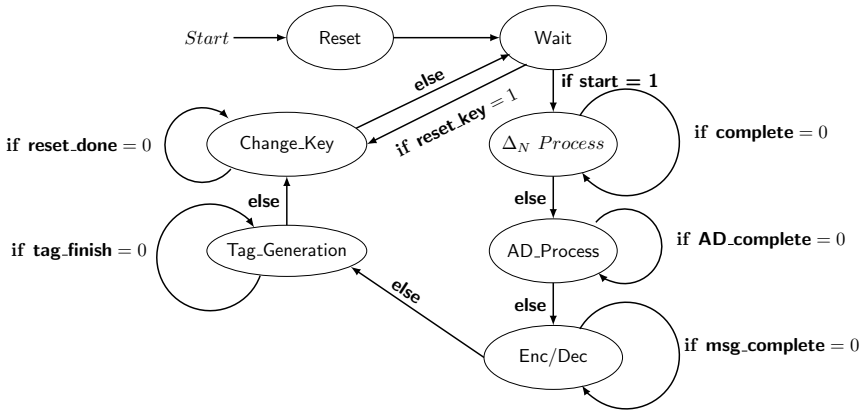

Figure 6: Finite State Machine

- Reset: This state resets all the internal variables and signals and prepares the circuit to start. The control from the Reset state goes to the Wait state.
- Wait: This state indicates that we should now initialize the cipher functionalities. It actually prepares the circuit to process the nonce. The control next enters into the state Δ_N Process state when the data signal start is set. Otherwise, it enters into the state Change_Key when the signal reset_key is set.
- Δ_N Process: This state initializes the cipher by computing Δ_N. When the computation is done (indicated by the complete signal), it enters into the associated data processing phase AD_Process. Otherwise, it will remain in the Δ_N Process state.
- Change_Key: This state indicates that the key is reset. The control transits to the state Wait when the reset is done. Otherwise, the control will remain int his state.

- AD_Process: This state indicates processing of the associated data. It internally uses the underlying block cipher, runs it and updates the intermediate checksum. The completion of this phase it indicated by the AD_complete signal. After the completion, the state transits to the Enc/Dec state which indicates the start of the message (or ciphertext) processing. Otherwise, it remains in the same state.

- Enc/Dec: This state indicates the message (or ciphertext) processing phase. It also invoke the block cipher internally, runs it and updates the checksum. The completion of this phase is indicated by the msg_complete signal. After the completion, the control enters into the Tag_Generation state. Otherwise, the control will remain in this state. Note that, during the decryption for LOCUS, the circuit runs the block cipher decryption module.

- Tag_Generation: This state indicates that the tag needs to be generated now. The completion of this state is indicated by the tag_finish signal and the control will go to the Wait state again. Otherwise, the control will remain in the same state when the tag_finish signal is not set.

5.2 Implementation of TweGIFT-64

In this section, we first briefly describe our hardware implementation details of the TweGIFT-64 module. We have implemented TweGIFT-64 using a basic iterative type architecture. We would like to emphasize that our implementation is round-based and it uses 64-bit data path, a smaller implementation can be obtained using smaller data paths 4-bit, 8-bit, 16-bit or even serialized implementations.

Table 3 provides the implementation details of TweGIFT-64 on Virtex 6. It is evident from the results that the difference in the number of LUTs is 119 (caused by the inclusion of the decryption rounds and the multiplexers to select the input to the state register). The difference in terms of the number of slices is about 36 such that one slice in Virtex 6 has 4 LUTs and 2 Flip-flops (depends how a design is optimized and placed by the Xilinx tools).

Table 3: TweGIFT-64 Implemented FPGA Results on Virtex 6

Mode	# Slice Registers	# LUTs	# Slices	Frequency (MHZ)	Gbps	Mbps/ LUT	Mbps/ Slice
Enc/dec	273	734	270	425.99	0.94	1.28	3.48
Enc	275	333	134	540.56	1.19	3.57	8.88

Table 4: TweGIFT-64 Implemented FPGA Results on Virtex 7

Platform	# Slice Registers	# LUTs	# Slices	Frequency (MHZ)	Gbps	Mbps/ LUT	Mbps/ Slice
Enc/dec	273	730	265	441.71	0.97	1.32	3.66
Enc	275	329	134	554.32	1.22	3.71	9.10

Detailed descriptions can be found in Appendix B.1.

5.3 Implementation of LOCUS and LOTUS

The hardware implementations of LOCUS and LOTUS use a round-based iterative TweGIFT-64 core as a main building block. Our designs are implemented optimizing the speed as the ones in the CAESAR benchmark. The inherent parallel characteristics of LOCUS and LOTUS allow the implementers to explore various other options such as pipeline or unrolling architectures. However, we do not use such optimizations here to ensure fair

comparison. The detailed implementation results are depicted in Table 5. The areas are provided in terms of the number of slice registers, slice LUTs and the number of occupied slices. Note that, we use less than 1% of the FPGA in our implementations: 0.18/0.24 (LOCUS/ LOTUS) in Virtex 6, 0.33/0.45 (LOCUS/ LOTUS) in Virtex 7. This is due to the fact that both the devices are high-end, and hence the available resources are huge.

Table 5: LOCUS and LOTUS (combined enc/dec circuit) Implemented FPGA Results.

Platform	Scheme	# Slice Registers	# LUTs	# Slices	Frequency (MHZ)	Throughput (Gbps)	Mbps/ LUT	Mbps/ Slice
Virtex 6	LOCUS	437	1146	418	348.67	0.39	0.34	0.94
Virtex 7	LOCUS	430	1154	439	392.20	0.44	0.38	1.00
Virtex 6	LOCUS-e	427	698	250	368.34	0.41	0.59	1.65
Virtex 7	LOCUS-e	424	704	272	406.84	0.46	0.65	1.68
Virtex 6	LOTUS	571	868	317	351.25	0.39	0.45	1.24
Virtex 7	LOTUS	565	865	317	424.45	0.48	0.55	1.50
Virtex 6	LOTUS-e	564	801	251	380.84	0.43	0.53	1.70
Virtex 7	LOTUS-e	564	800	249	414.42	0.47	0.58	1.87
Virtex 6	LOTUS-d	566	804	245	379.83	0.43	0.53	1.74
Virtex 7	LOTUS-d	563	791	254	418.91	0.47	0.59	1.85

5.4 Benchmarking LOCUS and LOTUS

In this section, we provide a benchmark of hardware implementation results for both LOCUS and LOTUS with the ATHENa listed results in [ATHb, ATHa] on both Virtex 6 and 7.

5.4.1 Benchmarking Methodology

We would like to point out that our implementation architecture is simply round-based with a 64-bit datapath. The implementation does not follow any optimization (e.g, 4-bit serialized implementation). Our main motivation is to provide a rough benchmark of our basic round-based implementation with the same for the other designs. Most of the implementation results (actually almost all) we use are round-based and taken from [ATHb, ATHa]. The results for COFB and Beetle are also round-based and taken from [CIMN17, CDNY18]. We clearly state that, we ignore the overheads to support the GMU API (COFB and Beetle have also been implemented without GMU API) and mention that supporting this API will add some overheads to the current figures of LOTUS and LOCUS. Nevertheless, we believe the current results of our designs suggest that low area hardware implementations are possible compared to other AE modes shown in the comparison tables, even if we add API overheads.
We also point out that it is basically a little hard to compare LOTUS and LOCUS using GIFT-64/128 with other 128-bit block cipher based or non block-cipher based AE schemes, because of the primitive differences and the types of security guarantees. For an instance, ACORN does not have any provable security result and has been analyzed with several cryptanalysis [DRA16, SWB+16, SBD+16, LLMH16]. DEOXYS, Joltik and JAMBU-SIMON employ lightweight (tweakable) blockciphers that allow different implementation size from GIFT-64/128. Sponge based AE schemes (Beetle , ASCON, Ketje, NORX, and PRIMATES-HANUMAN) use a public permutation with a large state size (greater than or equal to 256-bits) and avoid key scheduling circuit and have the provable security relying on the random permutation model.

A detailed comparison can be found below in Table 6 and 7. Note that, the hardware areas for SUNDAE [BBLT18] is given in GEs (ASIC platform). Hence, we do not include these results in the table. The comparison table shows that our implementation results are highly competitive. Among non-AES based constructions, LOCUS and LOTUS have

Table 6: Comparison on Virtex 6 [ATHb]. Here BC denotes block cipher, SC denotes Streamcipher, (T)BC denotes (Tweakable) block cipher and BC-RF denotes the block cipher's round function,'-' means that the data is not available.

Scheme	Underlying Primitive	# LUTs	# Slices	Gbps	Mbps/ LUT	Mbps/ Slice
LOCUS	BC (non AES)	1146	418	0.39	0.34	0.94
LOTUS	BC (non AES)	868	317	0.39	0.45	1.24
CLOC-TWINE [IMG+16]	BC (non-AES)	1689	532	0.343	0.203	0.645
SILC-LED [IMG+16]	BC (non-AES)	1685	579	0.245	0.145	0.422
SILC-PRESENT [IMG+16]	BC (non-AES)	1514	548	0.407	0.269	0.743
JAMBU-SIMON [WH16]	BC (non-AES)	1222	453	0.363	0.297	0.801
AES-OTR [Min16]	BC	5102	1385	2.741	0.537	1.979
AES-OCB [KR16]	BC	4249	1348	3.122	0.735	2.316
AES-OCB [KR16]	BC	4249	1348	1.56	0.37	1.16
AES-GCM [Dwo11]	BC	3175	1053	3.239	1.020	3.076
AES-COPA [ABL+15]	BC	7754	2358	2.500	0.322	1.060
CLOC-AES [IMG+16]	BC	3145	891	2.996	0.488	1.724
ELmD [DN15]	BC	4302	1584	3.168	0.736	2.091
JAMBU-AES [WH16]	BC	1836	652	1.999	1.089	3.067
SILC-AES [IMG+16]	BC	3066	921	4.040	1.318	4.387
COFB-AES [CIMN17, CIMN17]	BC	1075	442	2.850	2.240	6.450
AEGIS [WP16]	BC-RF	7592	2028	70.927	9.342	34.974
DEOXYS [JNP16]	TBC	3143	951	2.793	0.889	2.937
Beetle[Light+] [CDNY18]	Sponge	616	252	1.879	3.050	7.369
Beetle[Secure+] [CDNY18]	Sponge	998	434	2.520	2.525	5.806
ASCON-128 [DEMS16]	Sponge	1271	413	3.172	2.496	7.680
Ketje-Jr [BJDAK16]	Sponge	1236	412	2.832	2.292	6.875
NORX [AJN16]	Sponge	2964	1016	11.029	3.721	10.855
PRIMATES-HANUMAN [ABB+16]	Sponge	1012	390	0.964	0.953	2.472
ACORN [Wu16]	SC	455	135	3.112	6.840	23.052
TriviA-ck [CCHN15, CCHN18, CN15]	SC	2118	687	15.374	7.259	22.378

a better throughput/area metric. Moreover, our modes outperform AES-based parallel designs due to the smaller 64-bit TweGIFT block cipher.

6 Security Analysis of LOCUS and LOTUS

Before delving into the security proofs, we give an alternative formulation for LOCUS and LOTUS based on a tweakable block cipher. This formulation extends Rogaway's XEX [Rog04] based abstraction of OCB.

6.1 Θ-LOC and Θ-LOT

Let $\mathcal{T} = \{0,1\}^\kappa \times \{2,3,\ldots,15\} \times [2^n]$ and $\widetilde{\Pi} \leftarrow\!\!\$\ \mathsf{TPerms}(\mathcal{T}, \{0,1\}^n)$. We define two new authenticated encryption schemes $\Theta\text{-}\mathsf{LOT}[\widetilde{\Pi}]$ and $\Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}]$ in Algorithms 3 and 4, respectively.

Notice that the modified algorithms are implicitly keyed due to the tweakable random permutation $\widetilde{\Pi}$.

Let $\widetilde{\mathsf{E}}$ be a tweakable ideal cipher over key space $\{0,1\}^\kappa$, tweak space (15], and block space $\{0,1\}^n$. Now, we define $\widetilde{\mathsf{P}}$ as a tweakable block cipher over key space $\{0,1\}^\kappa$, tweak space $\mathcal{T}$, and block space $\{0,1\}^n$, by the following mapping:
$\forall (K, N, d, i, X) \in \{0,1\}^\kappa \times \mathcal{T} \times \{0,1\}^n$,

$$\widetilde{\mathsf{P}}_K^{N,d,i}(X) := \widetilde{\mathsf{E}}_{L_i}^d(X \oplus \Delta_N) \oplus \Delta_N. \tag{7}$$

where $L_i = 2^i(K \oplus N)$ and $\Delta_N = \widetilde{\mathsf{E}}_{K\oplus N}^1(\widetilde{\mathsf{E}}_K^0(0))$.

This definition, though artificial in nature, serves its purpose well. Notably, we can now view LOTUS and LOCUS as instantiations of Θ-LOT and Θ-LOC, namely, $\Theta\text{-}\mathsf{LOT}[\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]]$

Algorithm 3 The encryption algorithm of Θ-LOT$[\widetilde{\Pi}]$.

function Θ-LOT$[\widetilde{\Pi}]$.enc(N, A, M)
  $C \leftarrow \bot, W_\oplus \leftarrow 0, V_\oplus \leftarrow 0$
  $\lceil |A|/n \rceil = a$
  $\lceil |M|/n \rceil = m$
  if $a \neq 0$ **then**
    $V_\oplus \leftarrow$ proc_ad(A)
  if $m \neq 0$ **then**
    $(W_\oplus, C) \leftarrow$ proc_pt(M)
  $T \leftarrow$ proc_tg$(V_\oplus, W_\oplus)$
  return (C, T)

function proc_ad(A)
  $(A_1, \ldots, A_a) \xleftarrow{n} A$
  for $i = 1$ **to** $a - 1$ **do**
    $V_i \leftarrow \widetilde{\Pi}^{(N,2,i)}(A_i)$
    $V_\oplus \leftarrow V_\oplus \oplus V_i$
  if $|A_a = n|$ **then**
    $V_a \leftarrow \widetilde{\Pi}^{(N,2,a)}(\mathsf{ozs}(A_a))$
  else
    $V_a \leftarrow \widetilde{\Pi}^{(N,3,a)}(\mathsf{ozs}(A_a))$
  $V_\oplus \leftarrow V_\oplus \oplus V_a$
  return $V_\oplus$

function proc_pt(M)
  $(M_1, \ldots, M_m) \xleftarrow{n} M$
  $d = \lceil m/2 \rceil$
  for $i = 1$ **to** $d - 1$ **do**
    $j = 2i - 1$
    $W_j \leftarrow \widetilde{\Pi}^{(N,4,a+i)}(M_j)$
    $C_j \leftarrow \widetilde{\Pi}^{(N,7,a+i)}(W_j) \oplus M_{j+1}$
    $W_{j+1} \leftarrow \widetilde{\Pi}^{(N,5,a+i)}(C_j)$
    $C_{j+1} \leftarrow \widetilde{\Pi}^{(N,8,a+i)}(W_{j+1}) \oplus M_j$
    $W_\oplus \leftarrow W_\oplus \oplus W_1 \oplus W_2$
  $X \leftarrow \langle |M| - 2(d-1)n \rangle_n$
  $W_{2d-1} \leftarrow \widetilde{\Pi}^{N,12,a+d}(X)$
  $Y \leftarrow \widetilde{\Pi}^{N,14,a+d}(X) \oplus M_{2d-1}$
  $C_{2d-1} \leftarrow \lfloor Y \rfloor_{|M_{2d-1}|}$
  $W_\oplus \leftarrow W_\oplus \oplus W_{2d-1}$
  $C \leftarrow (C_1, \ldots, C_{2d-1})$
  if $2d = m$ **then**
    $W_{2d} \leftarrow \widetilde{\Pi}^{(N,13,a+d)}(Y)$
    $W_\oplus \leftarrow W_\oplus \oplus W_{2d}$
    $Y \leftarrow \lfloor \widetilde{\Pi}^{(N,15,a+d)}(W_{2d}) \oplus X \rfloor_{|M_{2d}|}$
    $C_{2d} \leftarrow Y \oplus M_{2d}$
    $C \leftarrow C \| C_{2d}$
  $W_\oplus \leftarrow W_\oplus \oplus M_m$
  return $(W_\oplus, C)$

function proc_tg$(V_\oplus, W_\oplus)$
  $X_\oplus \leftarrow V_\oplus \oplus W_\oplus$
  $T \leftarrow \widetilde{\Pi}^{N,6,a+m}(X_\oplus)$
  return T

Algorithm 4 The encryption algorithm of Θ-LOC$[\widetilde{\Pi}]$. The subroutine proc_ad and proc_tg are identical to the one used in Θ-LOT$[\widetilde{\Pi}]$.

function Θ-LOC$[\widetilde{\Pi}]$.enc(N, A, M)
  $C \leftarrow \bot,\ W_\oplus \leftarrow 0,\ V_\oplus \leftarrow 0$
  if $|A| \neq 0$ **then**
    $V_\oplus \leftarrow$ proc_ad(A)
  if $|M| \neq 0$ **then**
    $(W_\oplus, C) \leftarrow$ proc_pt(M)
  $T \leftarrow$ proc_tg$(V_\oplus, W_\oplus, |A| + |M|)$
  return (C, T)

function proc_pt(M)
  $(M_1, \ldots, M_m) \xleftarrow{n} M$
  for $j = 1$ **to** $m - 1$ **do**
    $W_j \leftarrow \widetilde{\Pi}^{(N,4,j)}(M_j)$
    $W_\oplus \leftarrow W_\oplus \oplus W_j$
    $C_j \leftarrow \widetilde{\Pi}^{(N,12,j)}(W_j)$
  $X \leftarrow \langle |M_m| \rangle_n$
  $W_m \leftarrow \widetilde{\Pi}^{(N,5,j)}(X)$
  $W_\oplus \leftarrow W_\oplus \oplus W_m \oplus M_m$
  $Y \leftarrow \widetilde{\Pi}^{(N,13,j)}(W_m)$
  $C_m \leftarrow \lfloor Y \rfloor_{|M_m|} \oplus M_m$
  $C \leftarrow (C_1, \ldots, C_m)$
  return $(W_\oplus, C)$

Table 7: Comparison on Virtex 7 [ATHb].

Scheme	# LUTs	# Slices	Gbps	Mbps/ LUT	Mbps/ Slice
LOCUS	1154	439	0.44	0.38	1.00
LOTUS	865	317	0.48	0.55	1.50
CLOC-TWINE	1552	439	0.432	0.278	0.984
SILC-AES	3040	910	4.365	1.436	4.796
SILC-LED	1682	524	0.267	0.159	0.510
SILC-PRESENT	1514	484	0.479	0.316	0.990
JAMBU-SIMON	1200	419	0.368	0.307	0.878
AES-OTR	4263	1204	3.187	0.748	2.647
OCB	4269	1228	3.608	0.845	2.889
AES-COPA	7795	2221	2.770	0.355	1.247
AES-GCM	3478	949	3.837	1.103	4.043
CLOC-AES	3552	1087	3.252	0.478	1.561
ELmD	4490	1306	4.025	0.896	3.082
JAMBU-AES	1595	457	1.824	1.144	3.991
COFB-AES	1456	555	2.820	2.220	5.080
SAEB [NMSS18]	348	–	–	–	–
AEGIS	7504	1983	94.208	12.554	47.508
DEOXYS	3234	954	1.472	0.455	2.981
Beetle[Light+]	608	312	2.095	3.445	6.715
Beetle[Secure+]	1101	512	2.993	2.718	5.846
ASCON-128	1373	401	3.852	2.806	9.606
Ketje-Jr	1567	518	4.080	2.604	7.876
NORX	2881	857	10.328	3.585	12.051
PRIMATES-HANUMAN	1148	370	1.072	0.934	2.897
ACORN	499	155	3.437	6.888	22.174
TriviA-ck	2221	684	14.852	6.687	21.713

and Θ-LOC$[\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]]$, respectively. We argue the security of LOCUS and LOTUS under this modified view.

We remark here that the small modifications in the specification of LOTUS and LOCUS (see section 3) are introduced precisely to exploit this modularity. As we see later in this section, these changes make the proof modular and much easier to understand. The security of the original construction as given in the NIST submission [CDJ+19a] is exactly the same, though requires a more dedicated and notationally complex proof.

6.2 Privacy and Integrity Security of LOCUS

Recall that LOCUS$[\widetilde{\mathsf{E}}]$ is equivalent to Θ-LOC$[\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]]$. As a result it is sufficient to bound the privacy and integrity-under-RUP advantage of Θ-LOC$[\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]]$. The main technical result on the security of Θ-LOC$[\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]]$ is given in Theorem 2.

Theorem 2. *For $\sigma_e + \sigma_d + \sigma_v \leq 2^{n-1}$, $q_p \leq 2^{\kappa-1}$, and $\sigma = \sigma_e + \sigma_d + \sigma_v$, we have*

1. *For all (q_e, σ_e, q_p)-distinguisher $\mathscr{D}$,*

$$\mathbf{Adv}^{\mathsf{priv}}_{\Theta\text{-LOC}[\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]]}(\mathscr{D}) \leq \frac{6q_p(q_e+\sigma_e)}{2^{n+\kappa}} + \frac{q_e^2+3\sigma_e^2}{2^{n+\kappa}} + \frac{q_p+q_e+\sigma_e}{2^{\kappa}}. \tag{8}$$

2. *For all $(q_e, q_v, \sigma_e, \sigma_d, \sigma_v, q_p)$-distinguisher $\mathscr{F}$,*

$$\mathbf{Adv}^{\mathsf{int\text{-}rup}}_{\Theta\text{-LOC}[\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]]}(\mathscr{F}) \leq \frac{6q_p(q+\sigma)}{2^{n+\kappa}} + \frac{q^2+3\sigma^2}{2^{n+\kappa}} + \frac{q_p+q+\sigma}{2^{\kappa}} + \frac{4q_v}{2^n}, \tag{9}$$

where $q = q_e + q_d + q_v$ and $\sigma = \sigma_e + \sigma_d + \sigma_v$.

Proof. As a first step, we replace $(\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}], \widetilde{\mathsf{E}})$ with $(\widetilde{\Pi}, \widetilde{\mathsf{E}})$ using a standard hybrid argument that incurs a cost of $\mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]}(\sigma_e + q_e, q_p)$ and $\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]}(\sigma + q, q_p)$ in case of privacy and integrity-under-RUP, respectively. The TSPRP advantage of $\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]$ is upper bounded in Lemma 1.

By a slight abuse of notation we reuse $\mathscr{D}$ as the distinguisher that tries to distinguish $\mathcal{P}_0 := (\Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{enc}, \widetilde{\mathsf{E}}^{\pm})$ and $\mathcal{P}_1 := (\$_{\mathsf{e}}, \widetilde{\mathsf{E}}^{\pm})$. Similarly, $\mathscr{F}$ denotes the distinguisher that tries to distinguish $\mathcal{R}_0 := (\Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{enc}, \Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{dec}, \Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{ver}, \widetilde{\mathsf{E}}^{\pm})$ and $\mathcal{R}_1 := (\Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{enc}, \Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{dec}, \perp, \widetilde{\mathsf{E}}^{\pm})$. The rest of the proof is sub-divided into two parts corresponding to Eq. (8) and Eq. (9).

Proof of privacy: We have

$$\begin{aligned}\Delta_{\mathscr{D}}[\mathcal{P}_0; \mathcal{P}_1] &= \left|\Pr[\mathscr{D}^{(\Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{enc}, \widetilde{\mathsf{E}}^{\pm})} = 1] - \Pr[\mathscr{D}^{(\$_{\mathsf{e}}, \widetilde{\mathsf{E}}^{\pm})} = 1]\right| \\ &= \left|\Pr[\mathscr{D}^{\Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{enc}} = 1] - \Pr[\mathscr{D}^{\$_{\mathsf{e}}} = 1]\right|. \end{aligned} \tag{10}$$

The second equality follows from the fact that access to $\widetilde{\mathsf{E}}^{\pm}$ does not help $\mathscr{D}$, as $\Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{enc}$ and $\$_{\mathsf{e}}$ are independent of $\widetilde{\mathsf{E}}$, whence the oracle can be dropped. We claim that the R.H.S. of Eq. (10) is 0. This can be argued based on two simple facts.

First, the output distribution of $\widetilde{\Pi}$ is identical to a uniform random function, if $\widetilde{\Pi}$ is always evaluated over distinct tweak values. This is independent of block cipher inputs as well as the actual tweak values. More formally, for all tweak values $x \in \mathcal{T}$, the mapping $\widetilde{\Pi}(x, \star)$ is identical to $\Gamma(x)$ where Γ denotes a uniform random function from $\mathcal{T}$ to $\{0,1\}^n$.

Second, in $\Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{enc}$, for nonce-respecting queries, $\widetilde{\Pi}$ is called exactly once for each tweak value. Hence the output distribution of $\Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{enc}$ is identical to $\$_{\mathsf{e}}$. The result follows by substituting the upper bound for $\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]}$ from Lemma 1.

Proof of integrity-under-RUP: Let $\mathcal{O}_0 := (\Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{enc}, \Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{dec}, \perp)$ and $\mathcal{O}_1 := (\Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{enc}, \Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{dec}, \Theta\text{-}\mathsf{LOC}[\widetilde{\Pi}].\mathsf{ver})$. Using a similar line of argument as in the case of Eq. (10), we have

$$\Delta_{\mathscr{F}}[\mathcal{R}_0; \mathcal{R}_1] = \left|\Pr[\mathscr{F}^{\mathcal{O}_0} = 1] - \Pr[\mathscr{F}^{\mathcal{O}_1} = 1]\right|. \tag{11}$$

Let $[q]$ denote the query indices, and $[q]_e$, $[q]_d$, and $[q]_v$ denote the subset of encryption, decryption, and verification query indices, respectively, i.e., $|[q]_x| = q_x$ for $x \in \{e, d, v\}$. All the encryption query variables (including the internal ones) are defined analogous to Algorithm 3 and Algorithm 4. The variables arising in decryption and verification queries are defined identically, but topped with tilde and bar, respectively.

Let Ω denote the set of attainable transcripts in the ideal world. For any transcript $\omega \in \Omega$, we segregate the encryption, decryption, and verification query tuples into ω^e, ω^d, and ω^v, i.e. $\omega^e = (N^i, A^i, M^i, C^i, T^i)_{i \in [q]_e}$, $\omega^d = (\widetilde{N}^i, A^i, \widetilde{C}^i, \widetilde{M}^i)_{i \in [q]_d}$, $\omega^v = (\bar{N}^i, \bar{A}^i, \bar{C}^i, \bar{T}^i, \perp^i)_{i \in [q]_v}$, and $\omega = \{\omega^e, \omega^d, \omega^v\}$.

We take all attainable transcripts to be good, i.e., $\Omega_{\mathtt{bad}} = \emptyset$. For a good transcript ω, it is obvious that $\Pr[\Lambda_1^e = \omega^e, \Lambda_1^d = \omega^d] = \Pr[\Lambda_0^e = \omega^e, \Lambda_0^d = \omega^d]$. Thus, the ratio of interpolation probabilities is given by

$$\begin{aligned}\frac{\Pr[\Lambda_1 = \omega]}{\Pr[\Lambda_0 = \omega]} &= \Pr[\Lambda_1^v = \omega^v | \Lambda_1^e = \omega^e, \Lambda_1^d = \omega^d] \\ &\geq 1 - \Pr[\Lambda_1^v \neq \omega^v | \Lambda_1^e = \omega^e, \Lambda_1^d = \omega^d],\end{aligned}$$

where we use the fact that $\Pr[\Lambda_0^v = \omega^v | \Lambda_0^e = \omega^e, \Lambda_0^d = \omega^d] = 1$. For $i \in [q]_v$, let $\mathtt{Forge}_i$ denote the event $(\bar{\mathsf{N}}^i, \bar{\mathsf{A}}^i, \bar{\mathsf{C}}^i, \bar{\mathsf{T}}^i, \bar{\lambda}^i) \neq (N^i, A^i, C^i, T^i, \perp) \mid \Lambda_1^e = \omega^e, \Lambda_1^d = \omega^d$, where $\bar{\lambda}^i$

denotes the output of the verification interface for the i-th verification query in the real oracle. Apart from $\bar{\lambda}^i$, all other variables are adversarial inputs, and hence must match. Then, we have

$$\Pr[\Lambda_1^v \neq \omega^v | \Lambda_1^e = \omega^e, \Lambda_1^d = \omega^d] \leq \sum_{i \in [q]_v} \Pr[\mathtt{Forge}_i].$$

We fix a verification query index i and follow the following two cases.

1. $\bar{\mathsf{N}}^i \neq \mathsf{N}^j$ for all $j \in [q]_e$. This means that in the real world, the tweakable random permutation $\widetilde{\Pi}$ was never called for tweak input $(\bar{\mathsf{N}}^i, 6, \cdot)$, whence the tag matches with at most 2^{-n} probability.

2. $\bar{\mathsf{N}}^i = \mathsf{N}^j$ for some $j \in [q]_e$. If $\bar{\mathsf{T}}^i \neq \mathsf{T}^j$, then the forgery succeeds with at most $1/(2^n - 1)$ probability, as this is equivalent of guessing the output of a uniform random permutation when one input-output pair is already known. Suppose $\bar{\mathsf{T}}^i = \mathsf{T}^j$. Then, we must have $\bar{\mathsf{X}}^i_{\oplus} = \mathsf{X}^j_{\oplus}$. Also, $(\bar{\mathsf{A}}^i, \bar{\mathsf{C}}^i) \neq (\mathsf{A}^j, \mathsf{C}^j)$, otherwise the queries are duplicate.

   We can have two cases, depending upon whether $\bar{\mathsf{A}}^j = \mathsf{A}^i$ or not. We discuss the $\bar{\mathsf{A}}^i = \mathsf{A}^j$, and the other case can be similarly bounded. Since $\bar{\mathsf{A}}^i = \mathsf{A}^j$, there must be at least one ciphertext block index, say k, in $[\max\{|\bar{\mathsf{C}}^i|, |\mathsf{C}^j|\}]$ such that $\bar{\mathsf{C}}^i_k \neq \mathsf{C}^j_k$. Now, we have two cases based on $|\bar{\mathsf{C}}^i|$ and $|\mathsf{C}^j|$.

   a. $|\bar{\mathsf{C}}^i| \neq |\mathsf{C}^j|$, say $|\bar{\mathsf{C}}^i| > |\mathsf{C}^j|$. Then, we choose $k = \bar{\ell}_i$. In this case, we condition on the values of $\bar{\mathsf{W}}^i$ and W^j as well as $\bar{\mathsf{V}}^i$ and V^j, except $\bar{\mathsf{W}}^i_{\bar{\ell}_i}$. Then, the probability that $\bar{\mathsf{X}}^i_{\oplus} = \mathsf{X}^j_{\oplus}$ is bounded by at most $1/(2^n - q_d - 1) < 2/2^n$ (assuming $q_d + 1 < 2^{n-1}$) due to the randomness of $\bar{\mathsf{W}}^i_{\bar{\ell}_i}$.

   b. $|\bar{\mathsf{C}}^i_k| = |\mathsf{C}^j_k|$. Suppose, the two ciphertexts differ only at the last block. Then it is easy to see that the probability of $\bar{\mathsf{X}}^i_{\oplus} = \mathsf{X}^j_{\oplus}$ is 0. This happens by design. Instead, suppose there exist $k < \ell_j$, such that $\bar{\mathsf{C}}^i_k \neq \mathsf{C}^j_k$. Then, the probability of $\bar{\mathsf{X}}^i_{\oplus} = \mathsf{X}^j_{\oplus}$ is bounded by $1/(2^n - q_d - 1) \leq 2/2^n$ (assuming $q_d + 1 < 2^{n-1}$), using a similar line of argument as in the preceding case.

   Cases 2a and 2b are mutually exclusive, which in combination with the $\bar{\mathsf{A}}^i \neq \mathsf{A}^j$ case, upper bounds the probability in case 2 by $4/2^n$.

Cases 1 and 2 are mutually exclusive, whence we can bound $\Pr[\mathtt{Forge}_i] \leq 4/2^n$. The result follows from Theorem 1. □

6.3 Privacy and Integrity Security of LOTUS

The main technical result on the security of LOTUS is given in Theorem 3.

Theorem 3. *For $\sigma_e + \sigma_d + \sigma_v \leq 2^{n-1}$, $q_p \leq 2^{\kappa-1}$, and $\sigma = \sigma_e + \sigma_d + \sigma_v$, we have*

1. *For all (q_e, σ_e, q_p)-distinguisher $\mathscr{D}$,*

$$\mathbf{Adv}^{\mathsf{priv}}_{\Theta\text{-}\mathsf{LOT}[\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]]}(\mathscr{D}) \leq \frac{6q_p(q_e + \sigma_e)}{2^{n+\kappa}} + \frac{q_e^2 + 3\sigma_e^2}{2^{n+\kappa}} + \frac{q_p + q_e + \sigma_e}{2^{\kappa}}. \tag{12}$$

2. *For all $(q_e, q_v, \sigma_e, \sigma_d, \sigma_v, q_p)$-distinguisher $\mathscr{F}$,*

$$\mathbf{Adv}^{\mathsf{int\text{-}rup}}_{\Theta\text{-}\mathsf{LOT}[\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]]}(\mathscr{F}) \leq \frac{6q_p(q + \sigma)}{2^{n+\kappa}} + \frac{q^2 + 3\sigma^2}{2^{n+\kappa}} + \frac{q_p + q + \sigma}{2^{\kappa}} + \frac{4q_v}{2^n}, \tag{13}$$

   where $q = q_e + q_d + q_v$ and $\sigma = \sigma_e + \sigma_d + \sigma_v$.

Proof. The privacy advantage is bounded to $\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]}(\sigma_e + q_e, q_p)$ using exactly the same argument as in case of Θ-LOC. The integrity-under-RUP advantage is bounded to $\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]}(\sigma + q, q_p) + 4q_v/2^n$ using similar arguments as in case of Θ-LOT. We skip a formal proof for economical reasons. □

6.4 Security of $\widetilde{\mathsf{P}}$

The main technical result on the security of $\widetilde{\mathsf{P}}$, as defined in section 6.1, is given in Lemma 1.

Lemma 1. *For any (q_e, q_d, q_p)-adversary $\mathscr{B}$, we have*

$$\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]}(\mathscr{B}) \leq \frac{q_p + q}{2^\kappa} + \frac{6q_p q}{2^{n+\kappa}} + \frac{q^2}{2^{n+\kappa}},$$

where $q = q_e + q_d$.

Proof. We employ the coefficient-H technique to bound the distinguishing advantage of $\mathscr{B}$ in distinguishing the real oracle $(\widetilde{\mathsf{P}}^{\pm}, \widetilde{\mathsf{E}}^{\pm})$ from the ideal oracle $(\widetilde{\Pi}^{\pm}, \widetilde{\mathsf{E}}^{\pm})$. Let $[q]$ denote the set of all construction query indices, and $[q]_e$, and $[q]_d$ denote the subset of encryption, and decryption, respectively, query indices, i.e., $|[q]_x| = q_x$ for $x \in \{e, d\}$.

For the i-th construction query, we define the following notations:

- $T_i := (N_i, d_i, m_i)$: the i-th tweak value; M_i: the i-th message; C_i: the i-th ciphertext.
- $K^i_N = K \oplus N_i$; $L_i := 2^{m_i} K^i_N$; $\Delta^i_N := \widetilde{\mathsf{E}}^1_{K^i_N}(\Delta_0)$, where $\Delta_0 = \widetilde{\mathsf{E}}^0_K(0)$.
- $X_i = M_i \oplus \Delta^i_N$; $Y_i = C_i \oplus \Delta^i_N$.

The i-th primitive query variables are defined analogously, but topped with a hat to differentiate them from their construction counterpart. So, the i-th primitive query is of the form $(\hat{L}_i, \hat{X}_i, \hat{Y}_i)$, where $\hat{L}_i$, $\hat{X}_i$, and $\hat{Y}_i$ denote the key, input and output of the primitive.

We consider an extended version of the oracles, in which they release the internal secrets, once the query-response phase is over. The real oracle releases the secret key K, and the Δ^i_{N} values for all $i \in [q]$. This uniquely defines all the intermediate variables arising in the construction queries.

The ideal oracle first samples a dummy key K uniformly at random. Let $\mathcal{S} = \{i \in [q] : \nexists j < i, \mathsf{N}_i = \mathsf{N}_j\}$. The ideal oracle samples Δ^i_{N} uniformly at random for all $i \in \mathcal{S}$, and sets $\Delta^j_{\mathsf{N}} = \Delta^i_{\mathsf{N}}$ if $\mathsf{N}_j = \mathsf{N}_i$ for all $j \in [q]$ and $i \in \mathcal{S}$. All other internal variables are defined according to their relationship in the real world.

Let Ω denote the set of attainable transcripts in the ideal world. For any transcript $\omega \in \Omega$, we segregate the construction and primitive query tuples into ω^c, and ω^p, i.e. $\omega^c = (N_i, d_i, m_i, M_i, C_i, X_i, Y_i, \Delta^i_N, K^i_N, L_i)_{i \in [q]}$, $\omega^p = (\hat{L}_i, \hat{d}_i, \hat{X}_i, \hat{Y}_i)_{i \in [q]_p}$.

Bad Transcript Analysis: We say that an attainable transcript is bad, if one of the following conditions hold:

C_0 : $\exists i \in [q]_p$ such that $K = \hat{L}_i$.

C_1 : $\exists i \in [q]$ such that $K = N_i$.

C_2 : $\exists i \in [q], j \in [q]_p$ such that $(\hat{L}_j, \hat{d}_j, \hat{Z}_j) = (K^i_N, 1, Z_i)$, where $(\hat{Z}_j, Z_i) \in \{(\hat{X}_j, \Delta_0), (\hat{Y}_j, \Delta^i_N)\}$.

C_3 : $\exists i \neq j \in [q]$ such that $(L_i, d_i, Z_i) = (L_j, d_j, Z_j)$, where $Z \in \{X, Y\}$.

$\mathtt{C}_4$: $\exists i \in [q], j \in [q]_p$ such that $(L_i, d_i, Z_i) = (\hat{L}_j, \hat{d}_j, \hat{Z}_j)$, where $Z \in \{X, Y\}$.

$\mathtt{C}_5$: $\exists i \in [q]$ such that $|\{j \in [q]_p : (\hat{L}_j, \hat{d}_j) = (L_i, d_i)\}| \geq 2^{n-1}$.

Let $\mathtt{bad}$ denote the event that Λ_0 satisfies one of the $\mathtt{C}_i$ for $i \in (5]$. Then, we have

$$\Pr[\Lambda_0 \in \Omega_{\mathtt{bad}}] = \Pr[\mathtt{bad}] = \Pr\left[\bigcup_{i=0}^{5} \mathtt{C}_i\right]. \tag{14}$$

It is easy to see that the probabilities $\Pr[\mathtt{C}_0]$ and $\Pr[\mathtt{C}_1]$ are bounded by at most $q_p 2^{-\kappa}$, and $q2^{-\kappa}$, respectively, since K is chosen uniformly at random. Now, we bound the probabilities of $\mathtt{C}_2$, $\mathtt{C}_3|\neg\mathtt{C}_1$, $\mathtt{C}_4$, and $\mathtt{C}_5$.

1. Bounding $\Pr[\mathtt{C}_2]$: In the ideal world, $\mathsf{K}_{\mathsf{N}}^i$, Δ_0, and Δ_{N}^i are all uniform and independent of each other. Further, there are two choices for $(\widehat{\mathsf{Z}}_j, \mathsf{Z}_i)$ and qq_p choices for i and j. So the probability of this event can be bounded by at most $qq_p 2^{1-n-\kappa}$.

2. Bounding $\Pr[\mathtt{C}_3|\neg\mathtt{C}_1]$: This event bounds the probability of internal key and input/output collision. Now we may have two cases:

(a) $\mathsf{N}_i = \mathsf{N}_j$. In this case, we must have $\mathsf{m}_i = \mathsf{m}_j$, otherwise $\mathsf{L}_i = 2^{\mathsf{m}_i}\mathsf{K}_{\mathsf{N}}^i \neq 2^{\mathsf{m}_j}\mathsf{K}_{\mathsf{N}}^i = \mathsf{L}_j$. Now $\mathsf{X}_i = \mathsf{X}_j$ implies that $\mathsf{M}_i = \mathsf{M}_j$ and $\mathsf{X}_i = \mathsf{X}_j$ implies that $\mathsf{C}_i = \mathsf{C}_j$, both of which imply duplicate query. So the probability is zero in this case.

(b) $\mathsf{N}_i \neq \mathsf{N}_j$. In this case, we have two equations $2^{\mathsf{m}_i}\mathsf{K}_{\mathsf{N}}^i = 2^{\mathsf{m}_j}\mathsf{K}_{\mathsf{N}}^j$ and $\mathsf{Z}_i = \mathsf{Z}_j$ in two independent random variables (K, and Δ_{N}^i) which gives a probability of $2^{-n-\kappa}$. Further, there are 2 choices for Z and $\binom{q}{2}$ choices for i and j. Thus, we have $\Pr[\mathtt{C}_3|\neg\mathtt{C}_1] \leq q^2 2^{-n-\kappa}$.

3. Bounding $\Pr[\mathtt{C}_4]$: This event is similar to 2.(b) above, and the probability can be bounded by at most $qq_p 2^{1-n-\kappa}$ using the randomness of K and Δ_{N}^i.

4. Bounding $\Pr[\mathtt{C}_5]$: This event is mainly useful in avoiding the case when the adversary accidentally exhausts the entire codebook for some construction query key L_i. Let $\widehat{\mathcal{K}}$ denote the set of all indices $i \in [q_p]$ such that $|\{j \in [q_p] : j > i, \widehat{\mathsf{L}}_j = \widehat{\mathsf{L}}_i\}| \geq 2^{n-1}$. Then $|\widehat{\mathcal{K}}| \leq q_p/2^{n-1}$. Since K is uniformly distributed, we have

$$\Pr[\mathtt{C}_5] = \Pr[\mathsf{L}_i \in \widehat{\mathcal{K}}] \leq \frac{2q_p q}{2^{n+\kappa}}.$$

On combining all bounds, we get

$$\Pr[\mathtt{bad}] \leq \frac{q_p + q}{2^{\kappa}} + \frac{6q_p q}{2^{n+\kappa}} + \frac{q^2}{2^{n+\kappa}}.$$

Good Transcript Analysis: Fix any good transcript ω. Let $(T'_1, \ldots, T'_r)$ denote the distinct tweaks present in $(T_1, \ldots, T_q)$. Let $(c_1, \ldots, c_r)$ be a tuple of positive integers with $c_i = |\{j \in [q] : T_j = T'_i\}|$. Clearly, $\sum_j c_j = q$ since the transcript is good (i.e. $(T_i, X_i) = (T_j, X_j) \iff (T_i, Y_i) = (T_j, Y_j)$). Now, in the ideal world we have

$$\begin{aligned}\Pr[\Lambda_0 = \omega] &= \Pr[\Lambda_0^p = \omega^p, \Lambda_0^c = \omega^c] \\ &= \Pr[\Lambda_0^p = \omega^p] \cdot \frac{1}{2^{\kappa} 2^{n(q+1)} \prod_{i=1}^{r} (2^n)_{c_i}}\end{aligned} \tag{15}$$

Let $((L'_1, d'_1), \ldots, (L'_s, d'_s))$ denote the distinct keys and short tweak tuples present in $((L_1, d_1), \ldots, (L_q, d_q))$. Let $(a_1, \ldots, a_s)$ and $(b_1, \ldots, b_s)$ be tuples of positive integers such

that $a_i = |\{j \in [q] : (L_j, d_j) = (L'_i, d'_i)\}|$ and $b_i = |\{j \in [q]_p : (\hat{L}_j, \hat{d}_j) = (L'_i, d'_i)\}|$. Clearly, $\sum_{i=1}^{s} a_i = q$, and $b_i < 2^{n-1}$ for all $i \in [s]$, since the transcript is good. Then, we have

$$\begin{aligned}\Pr[\Lambda_1 = \omega] &= \Pr[\Lambda_1^p = \omega^p] \cdot \Pr[\Lambda_1^c = \omega^c | \Lambda_1^p = \omega^p] \\ &= \Pr[\Lambda_0^p = \omega^p] \cdot \frac{1}{2^{\kappa} 2^{n(q+1)} \prod_{i=1}^{s} (2^n - b_i)_{a_i}} \end{aligned} \tag{16}$$

On dividing Eq. (16) by (15), and doing some simple algebraic simplifications, we get

$$\frac{\Pr[\Lambda_1 = \omega]}{\Pr[\Lambda_0 = \omega]} \geq 1.$$

The result follows from the coefficient-H technique. □

6.5 Some Remarks on Generic Cryptanalysis on LOCUS and LOTUS

Here we summarize some generic ways of attacking LOCUS. Similar strategies work for LOTUS as well. First of all it is clear from the proof of lemma 1 that privacy directly depends on the security of $\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]$. Below, we enumerate some of the important attack strategies against $\widetilde{\mathsf{P}}[\widetilde{\mathsf{E}}]$:

1. Guessing master key by making primitive queries: One can exhaustively search for the master key which requires $q_p = O(2^{\kappa})$ and $q = O(1)$. This strategy is handled in event $\mathtt{C}_0$.

2. Guessing the nonce-based internal key and input mask: This requires a correct guess of $\mathsf{K} \oplus \mathsf{N}$ and $\widetilde{\mathsf{E}}^0_{\mathsf{K}}(0)$, which requires $q_p q = O(2^{n+\kappa})$. This strategy is handled in event $\mathtt{C}_2$.

3. Colliding the internal key and input of two distinct $\widetilde{\mathsf{P}}$ queries: Clearly, this requires $q = O(2^{\frac{n+\kappa}{2}})$, as both key and input should collide. We remark that similar event requires just $q = O(2^{n/2})$ queries in XEX based constructions. For instance, Ferguson's forgery attack [Fer02] creates a collision on the internal input in $O(2^{n/2})$ queries. However, the same attack does not succeed against LOCUS, as just internal input collision is not enough. This strategy is handled in event $\mathtt{C}_3$.

In INT-RUP attacks the adversary can either try to exploit the above mentioned attacks, or it can try to guess the tag or try to collide the internal checksum values. All these cases are handled in the proof of Theorem 2. Note that the access to unverified plaintext gives no extra advantage to the adversary, as the plaintext is not used in the checksum computation.

We also remark that the recent attacks on OCB2 by Inoue et al. [IIMP19] is also not applicable against LOCUS. Basically, their attack exploits a flaw in the last block processing of OCB2. Both LOCUS and LOTUS are devoid of such flaws.

Acknowledgments

The authors would like to thank Dr. Nicky Mouha for his insightful comments and suggestions in preparing the final draft. We would also like to thank all the anonymous reviewers of ToSC 2019 for their valuable comments. Nilanjan Datta, Ashwin Jha and Mridul Nandi are supported by the project "Study and Analysis of IoT Security" under Government of India at R.C.Bose Centre for Cryptology and Security, Indian Statistical Institute, Kolkata.

References

[ABB+16] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. PRIMATEs v1.02. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round2/primatesv102.pdf.

[ABL+14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 105–125, 2014.

[ABL+15] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. AES-COPA v.2. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/aescopav2.pdf.

[AJN16] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v3.0. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/norxv30.pdf.

[ATHa] ATHENa: Automated Tool for Hardware Evaluation. https://cryptography.gmu.edu/athena.

[ATHb] Authenticated Encryption FPGA Ranking. https://cryptography.gmu.edu/athenadb/fpga_auth_cipher/rankings_view.

[BBLT18] Subhadeep Banik, Andrey Bogdanov, Atul Luykx, and Elmar Tischhauser. Sundae: Small universal deterministic authenticated encryption for the internet of things. *IACR Transactions on Symmetric Cryptology*, 2018(3):1–35, Sep. 2018.

[BBM15] Subhadeep Banik, Andrey Bogdanov, and Kazuhiko Minematsu. Low-Area Hardware Implementations of CLOC, SILC and AES-OTR, 2015. DIAC 2015.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *Selected Areas in Cryptography - 18th International Workshop, SAC 2011. Revised Selected Papers*, pages 320–337, 2011.

[BHT18] Priyanka Bose, Viet Tung Hoang, and Stefano Tessaro. Revisiting AES-GCM-SIV: multi-user security, faster key derivation, and better bounds. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, pages 468–499, 2018.

[BJDAK16] Guido Bertoni, Michaël Peeters Joan Daemen, Gilles Van Assche, and Ronny Van Keer. Ketje v2. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/ketjev2.pdf.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 123–153, 2016.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES 2007*, pages 450–466, 2007.

[BKR98] Mihir Bellare, Ted Krovetz, and Phillip Rogaway. Luby-rackoff backwards: Increasing security by making block ciphers non-invertible. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, pages 266–280, 1998.

[BMR+13] Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: AES-Based Lightweight Authenticated Encryption. In *FSE 2013*, pages 447–466, 2013.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 321–345, 2017.

[BT16] Mihir Bellare and Björn Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 247–276, 2016.

[CAE14] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness, 2014. http://competitions.cr.yp.to/caesar.html.

[CCHN15] Avik Chakraborti, Anupam Chattopadhyay, Muhammad Hassan, and Mridul Nandi. Trivia: A fast and secure authenticated encryption scheme. In *CHES 2015*, pages 330–353, 2015.

[CCHN18] Avik Chakraborti, Anupam Chattopadhyay, Muhammad Hassan, and Mridul Nandi. Trivia and utrivia: two fast and secure authenticated encryption schemes. *J. Cryptographic Engineering*, 8(1):29–48, 2018.

[CDJ+19a] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancilias-López, Mridul Nandi, and Yu Sasaki. LOTUS-AEAD and LOCUS-AEAD. Submission to NIST LwC Standardization Process (Round 1), 2019.

[CDJ+19b] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas-López, Mridul Nandi, and Yu Sasaki. Elastic-tweak: A framework for short tweak tweakable block cipher. *IACR Cryptology ePrint Archive*, 2019:440, 2019.

[CDNY18] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle family of lightweight and secure authenticated encryption ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):218–241, 2018.

[CIMN17] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? In *CHES 2017*, pages 277–298, 2017.

[CN15] Avik Chakraborti and Mridul Nandi. TriviA-ck-v2. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/triviackv2.pdf.

[CWZ19] Lele Chen, Gaoli Wang, and GuoYan Zhang. MILP-based Related-Key Rectangle Attack and Its Application to GIFT, Khudra, MIBS. *The Computer Journal*, 10 2019. bxz076.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/asconv12.pdf.

[DN15] Nilanjan Datta and Mridul Nandi. Proposal of ELmD v2.1. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/elmdv21.pdf.

[DRA16] Prakash Dey, Raghvendra Singh Rohit, and Avishek Adhikari. Full key recovery of ACORN with a single fault. *J. Inf. Sec. Appl.*, 29:57–64, 2016.

[Dwo11] Morris Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC, NIST Special Publication 800-38D, 2011. csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

[Fer02] Niels Ferguson. Collision attacks on ocb, 2002. Online: https://web.cs.ucdavis.edu/~rogaway/ocb/fe02.pdf.

[FJMV03] Pierre-Alain Fouque, Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. Authenticated on-line encryption. In *Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003, Revised Papers*, pages 145–159, 2003.

[GPR14] Peter Gazi, Krzysztof Pietrzak, and Michal Rybár. The exact prf-security of NMAC and HMAC. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 113–130, 2014.

[IIMP19] Akiko Inoue, Tetsu Iwata, Kazuhiko Minematsu, and Bertram Poettering. Cryptanalysis of OCB2: attacks on authenticity and confidentiality. In *Advances in Cryptology - CRYPTO 2019, Proceedings, Part I*, pages 3–31, 2019.

[IMG+16] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. CLOC and SILC. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/clocsilcv3.pdf.

[JNP16] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Deoxys v1.41. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/deoxysv141.pdf.

[JZD19] Fulei Ji, Wentao Zhang, and Tianyou Ding. Improving matsui's search algorithm for the best differential/linear trails and its applications for des, DESL and GIFT. *IACR Cryptology ePrint Archive*, 2019:1190, 2019.

[KR16] Ted Krovetz and Phillip Rogaway. OCB(v1.1). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/ocbv11.pdf.

[LLMH16] Frédéric Lafitte, Liran Lerman, Olivier Markowitch, and Dirk Van Heule. SAT-based cryptanalysis of ACORN. *IACR Cryptology ePrint Archive*, 2016:521, 2016.

[LR85] Michael Luby and Charles Rackoff. How to construct pseudo-random permutations from pseudo-random functions (abstract). In *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, page 447, 1985.

[LS19] Yunwen Liu and Yu Sasaki. Related-key boomerang attacks on GIFT with automated trail search including BCT effect. In *ACISP 2019*, pages 555–572, 2019.

[MBTM17] Kerry A. McKay, Larry Bassham, Meltem Sönmez Turan, and Nicky Mouha. Report on Lightweight Cryptography, 2017. http://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf.

[Men17] Bart Mennink. Insuperability of the standard versus ideal model gap for tweakable blockcipher security. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 708–732, 2017.

[Min16] Kazuhiko Minematsu. AES-OTR v3.1. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/aesotrv31.pdf.

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *EUROCRYPT 2011*, pages 69–88, 2011.

[Nai17] Yusuke Naito. Tweakable blockciphers for efficient authenticated encryptions with beyond the birthday-bound security. *IACR Trans. Symmetric Cryptol.*, 2017(2):1–26, 2017.

[NMSS18] Yusuke Naito, Mitsuru Matsui, Takeshi Sugawara, and Daisuke Suzuki. SAEB: A lightweight blockcipher-based AEAD mode of operation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):192–217, 2018.

[OS19] OMA-SpecWorks. Lightweight-M2M, 2019. https://www.omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/.

[RBB03] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.

[Rog04] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, pages 16–31, 2004.

[SBD+16] Md. Iftekhar Salam, Harry Bartlett, Ed Dawson, Josef Pieprzyk, Leonie Simpson, and Kenneth Koon-Ho Wong. Investigating cube attacks on the authenticated encryption stream cipher ACORN. In *ATIS 2016*, pages 15–26, 2016.

[SWB+16] Md. Iftekhar Salam, Kenneth Koon-Ho Wong, Harry Bartlett, Leonie Ruth Simpson, Ed Dawson, and Josef Pieprzyk. Finding state collisions in the authenticated encryption stream cipher ACORN. In *Proceedings of the Australasian Computer Science Week Multiconference*, page 36, 2016.

[Vau03] Serge Vaudenay. Decorrelation: A Theory for Block Cipher Security. *J. Cryptology*, 16(4):249–286, 2003.

[WH16] Hongjun Wu and Tao Huang. The JAMBU Lightweight Authentication Encryption Mode (v2.1). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/jambuv21.pdf.

[WP16] Hongjun Wu and Bart Preneel. AEGIS : A Fast Authenticated Encryption Algorithm (v1.1). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/aegisv11.pdf.

[Wu16] Hongjun Wu. ACORN: A Lightweight Authenticated Cipher (v3). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/acornv3.pdf.

[ZDM+19] Boxin Zhao, Xiaoyang Dong, Willi Meier, Keting Jia, and Gaoli Wang. Generalized related-key rectangle attacks on block ciphers with linear key schedule. *IACR Cryptology ePrint Archive*, 2019:714, 2019.

[ZDY18] Baoyu Zhu, Xiaoyang Dong, and Hongbo Yu. Milp-based differential attack on round-reduced gift. Cryptology ePrint Archive, Report 2018/390, 2018. https://eprint.iacr.org/2018/390.

[ZWH17] Ping Zhang, Peng Wang, and Honggang Hu. The INT-RUP Security of OCB with Intermediate (Parity) Checksum. *IACR Cryptology ePrint Archive*, 2017. https://eprint.iacr.org/2016/1059.pdf.

[ZZDX19] Chunning Zhou, Wentao Zhang, Tianyou Ding, and Zejun Xiang. Improving the milp-based security evaluation algorithms against differential cryptanalysis using divide-and-conquer approach. Cryptology ePrint Archive, Report 2019/019, 2019. https://eprint.iacr.org/2019/019.

A Verified Decryption Algorithm

A.1 Verified Decryption Algorithm for LOCUS

The verified decryption algorithm for LOCUS is given in Algorithm 5.

A.2 Verified Decryption Algorithm for LOTUS

The verified decryption algorithm for LOTUS is given in Algorithm 6.

B Hardware Implementation

B.1 Hardware Implementation of TweGIFT-64

We first briefly describe our round based hardware implementation details of TweGIFT-64. The architecture described in Fig. 7 follows a simple base implementation. S-boxes are implemented as LUTs, as we follow the round based strategy, 16 S-boxes are implemented in one clock cycle. The bit-wise permutation is just in routing process. the round key addition (denoted by $Addrk$) selects 32 bits from the 128 bit key register and are added to the state along with the round constants. The three transformations explained above are implemented in a combinatorial fashion and the only synchronous component in the datapath is the state register. $muxIn$ selects the input Pt (input data block) when a new encryption or decryption starts (in the first round) or is feedbacked from the state register for the rest of the 27 rounds. The key generation uses an 128 bit shift register. Round constants are computed using a small 6 bit shift register. Both the procedures follow the

Algorithm 5 The verified decryption algorithm of LOCUS. The subroutines proc_ad and proc_tag are given in the encryption algorithm.

function LOCUS-AEAD_$\widetilde{\mathsf{E}}$.dec(K, N, A, C, T)
  $M \leftarrow \perp,\ W_\oplus \leftarrow 0,\ V_\oplus \leftarrow 0$
  $(K_N, \Delta_N) \leftarrow \mathsf{init}(K, N)$
  if $|A| \neq 0$ **then**
    $(K_N, V_\oplus) \leftarrow \mathsf{proc\_ad}(K_N, \Delta_N, A)$
  if $|M| \neq 0$ **then**
    $(K_N, W_\oplus, M) \leftarrow \mathsf{proc\_ct}(K_N, \Delta_N, C)$
  $T' \leftarrow \mathsf{proc\_tg}(K_N, \Delta_N, V_\oplus, W_\oplus)$
  if $T' = T$ **then**
    return M
  else
    return $\perp$

function proc_ct(K_N, Δ_N, A, C, T)
  $L \leftarrow K_N$
  $(C_1, \ldots, C_m) \xleftarrow{n} C$
  for $j = 1$ **to** $m - 1$ **do**
    $Y \leftarrow C_j \oplus \Delta_N$
    $L \leftarrow L \odot \alpha$
    $W \leftarrow \widetilde{\mathsf{E}}^{-1}_{L,12}(Y)$
    $W_\oplus \leftarrow W_\oplus \oplus W$
    $X \leftarrow \widetilde{\mathsf{E}}^{-1}_{L,4}(W)$
    $M_j \leftarrow X \oplus \Delta_N$
  $L \leftarrow L \odot \alpha$
  $X \leftarrow \langle |C_m| \rangle_n \oplus \Delta_N$
  $W \leftarrow \widetilde{\mathsf{E}}_{L,5}(X)$
  $Y \leftarrow \widetilde{\mathsf{E}}_{L,13}(W)$
  $M_m \leftarrow \mathsf{chop}(Y \oplus \Delta_N, |C_m|) \oplus C_m$
  $W_\oplus \leftarrow W_\oplus \oplus W \oplus M_m$
  $M \leftarrow (M_1, \ldots, M_m)$
  return $(L, W_\oplus, M)$

original specification of the GIFT-64-128 block cipher. As the key schedule operations contains only bit shifts and circular rotations, it is easy to get the round key $K_2 8$ from original key K_1 and vice-versa using the permutations showed in Table 8 and 9 respectively. Note that, depending on the context, we use "block cipher" to denote "tweakable block cipher".

The control flow is generated by a small finite state machine with three states: BC_Reset, BC_Wait, BC_Encrypt. BC_Reset initializes the key register with the key through the key port and then goes to BC_Wait until the start signal in activated. The BC_Encrypt state executes the block cipher rounds and after executing all the 28 rounds it returns to BC_Wait.

We optimize our TweGIFT-64 implementation to encrypt bulk information in the ECB mode. If the signal start is set to 1, an additional clock cycle for the initialization phase can be avoided. In addition, when the encryption of the actual state is performed, the input can be taken directly from the feedback using the multiplexer $muxIn$. These two optimizations allow us to save 1 clock cycle for each of the processed blocks.

For a single chip implementation, the multiplexer $muxSt$ selects the input to be send the state register from a decryption or an encryption round. For the encryption only module, all the decryption rounds and the multiplexer $muxSt$ are removed from the architecture and the encryption round is then connected directly to the state register.

B.2 Component Wise Area Calculation for lightweight LOCUS and LOTUS

Both the architectures of lightweight LOCUS and LOTUS function use several modules. Here, we provide a brief discussion about the distribution of the hardware footprints among the individual modules such as the main module, control unit module, block cipher round module, and the logic (means additional register, multiplexers, etc) components. The area utilization by the modules have been measured on Virtex 6. The distributions of the

Algorithm 6 The verified decryption algorithm of LOTUS.

function LOTUS-AEAD_$\widetilde{\mathsf{E}}$.dec(K, N, A, C, T)
  $M \leftarrow \perp,\ W_\oplus \leftarrow 0,\ V_\oplus \leftarrow 0$
  $(K_N, \Delta_N) \leftarrow \mathsf{init}(K, N)$
  if $|A| \neq 0$ **then**
    $(K_N, V_\oplus) \leftarrow \mathsf{proc\_ad}(K_N, \Delta_N, A)$
  if $|M| \neq 0$ **then**
    $(K_N, W_\oplus, M) \leftarrow \mathsf{proc\_ct}(K_N, \Delta_N, C)$
  $T' \leftarrow \mathsf{proc\_tg}(K_N, \Delta_N, V_\oplus, W_\oplus)$
  if $T' = T$ **then**
    return M
  else
    return $\perp$

function proc_ct(K_N, Δ_N, C)
  $L \leftarrow K_N$
  $(C_1, \ldots, C_m) \xleftarrow{n} C$
  $d = \lceil m/2 \rceil$
  for $i = 1$ **to** $d - 1$ **do**
    $j = 2i$
    $L \leftarrow L \odot \alpha$
    $X_1 \leftarrow C_j \oplus \Delta_N$
    $W_1 \leftarrow \widetilde{\mathsf{E}}_{L,8}(X_1)$
    $Y_1 \leftarrow \widetilde{\mathsf{E}}_{L,5}(W_1)$
    $X_2 \leftarrow C_{j+1} \oplus Y_1$
    $W_2 \leftarrow \widetilde{\mathsf{E}}_{L,4}(X_2)$
    $Y_2 \leftarrow \widetilde{\mathsf{E}}_{L,7}(W_2)$
    $W_\oplus \leftarrow W_\oplus \oplus W_1 \oplus W_2$
    $M_j \leftarrow X_2 \oplus \Delta_N$
    $M_{j+1} \leftarrow Y_2 \oplus X_1$
  $X_1 \leftarrow \langle |C| - 2(d-1)n \rangle_n \oplus \Delta_N$
  $L \leftarrow L \odot \alpha$
  $W_1 \leftarrow \widetilde{\mathsf{E}}_{L,12}(X_1)$
  $Y_1 \leftarrow \widetilde{\mathsf{E}}_{L,14}(W_1)$
  $M_{2d-2} \leftarrow \mathsf{chop}(Y_1 \oplus \Delta_N, |C_{2d-2}|) \oplus C_{2d-2}$
  $X_2 \leftarrow Y_1 \oplus M_{2d-2}$
  $W_\oplus \leftarrow W_\oplus \oplus W_1$
  $M \leftarrow (M_1, \ldots, M_{2d-1})$
  if $2d = m$ **then**
    $W_2 \leftarrow \widetilde{\mathsf{E}}_{L,13}(X_2)$
    $W_\oplus \leftarrow W_\oplus \oplus W_2$
    $Y_2 \leftarrow \widetilde{\mathsf{E}}_{L,15}(W_2)$
    $M_{2d} \leftarrow \mathsf{chop}(X_1 \oplus Y_2, |C_{2d}|) \oplus C_{2d}$
    $M \leftarrow M \| M_{2d}$
  $W_\oplus \leftarrow W_\oplus \oplus M_m$
  return $(L, W_\oplus, M)$

Table 8: Permutation to get K_1 from K_{28}

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
44	45	46	47	32	33	34	35	36	37	38	39	40	41	42	43
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
50	51	52	53	54	55	56	57	58	59	60	61	62	63	48	49
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
76	77	78	79	64	65	66	67	68	69	70	71	72	73	74	75
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
82	83	84	85	86	87	88	89	90	91	92	93	94	95	80	81
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
108	109	110	111	96	97	98	99	100	101	102	103	104	105	106	107
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
114	115	116	117	118	119	120	121	122	123	124	125	126	127	112	113
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
20	21	22	23	24	25	26	27	28	29	30	31	16	17	18	19

Table 9: Permutation to get original K_{28} from K_1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
104	105	106	107	108	109	110	111	96	97	98	99	100	101	102	103
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
124	125	126	127	112	113	114	115	116	117	118	119	120	121	122	123
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
30	31	16	17	18	19	20	21	22	23	24	25	26	27	28	29
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
36	37	38	39	40	41	42	43	44	45	46	47	32	33	34	35
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
62	63	48	49	50	51	52	53	54	55	56	57	58	59	60	61
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
68	69	70	71	72	73	74	75	76	77	78	79	64	65	66	67
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
94	95	80	81	82	83	84	85	86	87	88	89	90	91	92	93

areas are presented in terms of the number of LUTs for both LOCUS and LOTUS. We observe that, the majority of the hardware area have been consumed by the underlying block cipher. The distributions are described in Fig. 8 below.

In this section, we provide the hardware implementation details of both LOCUS and LOTUS with the underlying block cipher TweGIFT-64. In several applications, cipher implementations with small size are desirable. We primarily target these applications and implement the cipher with small hardware area. It is easy to see that both our designs share the same structure for the associated data processing, while they differ in the message processing phase.Both LOCUS and LOTUS have simple structure: they consist of a block cipher and a few basic operations (such as bitwise XORs, multiplexers and one accumulator). We would like to point out that, majority of the hardware areas are dominated by the TweGIFT-64 module. We describe the hardware architectures as well as provide our hardware implementation results on both Virtex 6 and 7.

We provide a brief analysis on clock cycles per byte (*cpb*). This is a theoretical way to estimate the speed of the architecture. We would like to note that both the designs shares the same values for *cpb* and we provide a joint analysis here. We consider round based architecture with 64 bit datapath. To process an associated data of a blocks and a message of m blocks, we need $29a + 57m$ clock cycles. We use one TweGIFT-64 call to process one associated data block and two TweGIFT-64 calls to process one message block. Our block cipher is optimized to process a bulk data, and the reset is required only to indicate that the stream processing starts. We observe that the *cpb* values for different a and m are constant as there is no initialization overhead and the overhead for the tag generation (constant small number of clock cycles) are negligible for long messages. Our

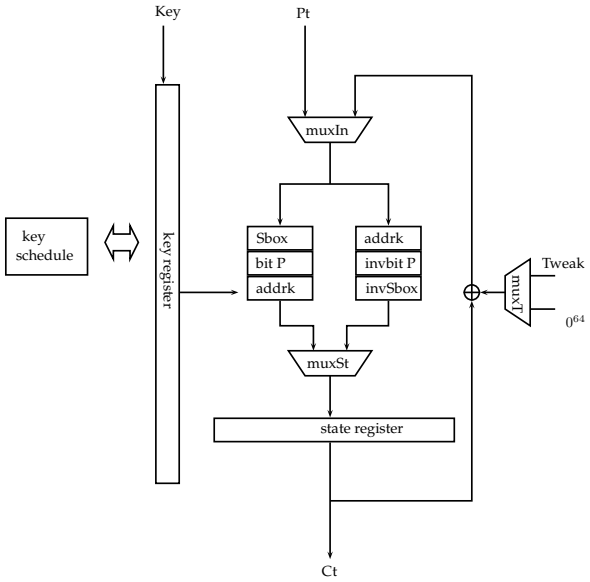

Figure 7: Architecture of TweGIFT-64 Implementation

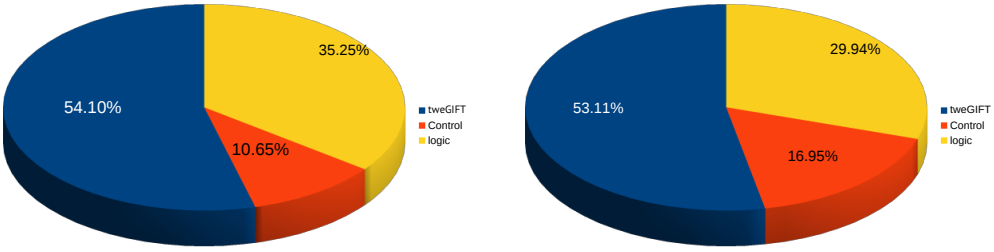

Figure 8: Distribution of Components by #LUTs for LOCUS (left) and LOTUS (right)

design accept 64 bit or 8 byte data blocks and the cpb is $(29a + 57m)/8m$. Assuming $a = m$, we have a constant value of cpb that equals to $60/8 = 10.75$.